UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO DE INGENIERÍA EN INFORMÁTICA

TRABAJO FINAL DE GRADO

Aplicación Android para el reconocimiento automático de actividades físicas en tiempo real

Autor: **David Martín de Castro**

Tutor: **Francisco Javier Ordóñez Morales**

Director: **Andrés Duque Fernández**

Leganés, 05 de septiembre de 2012

AGRADECIMIENTOS

Cuatro años. Para algunos serán pocos, para otros una barbaridad. Yo lo catalogo como una etapa en la que he tenido momentos buenos y momentos malos. Y es que el tiempo vuela o parece que se detiene. Si bien es cierto que aún me queda un camino por delante en mi etapa de aprendizaje, me gusta recordar a aquellas personas con las que, sin ellos, esto no habría sido lo mismo.

Comenzaré con mis padres. Esos padres que, aunque a nosotros no nos lo parezca, siempre hacen las mejores cosas para sus hijos. Después de casi 22 años juntos me conocéis perfectamente, sabéis lo que necesito y siempre estáis ahí para ayudarme y apoyarme. No soy perfecto, pero hago lo que puedo ;) Goyo, Nieves, muchas gracias por todo este tiempo y por todo lo que habéis hecho por mí. Jamás lo olvidaré. Te quiero papá. Te quiero mamá.

Aunque no esté en mi primer párrafo, no le doy menos importancia a mi preciosa novia Sara. Gracias a esta universidad y a la gente que conocí aquí, tú y yo hemos acabado juntos. Me has apoyado en lo bueno y en lo malo, en la salud y en la enfermedad, en la pobreza y en la gran pobreza XD Pero sé que sin ti esto no sería ni parecido de lo que es. Estoy muy feliz de estar contigo y me encantaría que este amor durara para siempre. Algunos me tacharán de ñoño (¡envidiosos!) pero es lo que realmente siento. Muchísimas gracias por estar a mi lado y aguantar mi frikismo (como dices tú :P). Te quiero amore!!!

Por supuesto, se merecen una buena mención Javier y Andrés, esos tutores que te apoyan, te ayudan, te enseñan y te entienden. Habéis sido la leche, no solo al ofrecerme el proyecto, sino al conseguir que se hiciera realidad. Reconozco que me lo he pasado genial con este TFG y repetiría. Muchas gracias chicos ;)

Como olvidar a mi familia, esa que siempre te quiere ayudar. Sois demasiados y no puedo extenderme mucho con este punto, así que muchísimas gracias a tod@s, tanto los de aquí como los de allí, españoles, ingleses, portugueses… Os quiero!

Ahora viene un grupo de personas que, antes o después, me habéis alegrado mucho la vida y hemos sufrido demasiado con esta nuestra universidad. Manicomio, 100Montaditos y el 2.2.C-Pasillo son unos de los sitios en los que más hemos estado. Me refiero, por supuesto, a Ana, Lore, Mario, Sandra y Víctor (grupo 100M), grandes amigos con los que me he reído hasta reventar y he sufrido en la universidad. Y lo que nos queda :'(Gracias por todo chic@s!

Por supuesto no me olvido del resto de gente que en la uni he conocido y que, gracias a esta universidad, he tenido el placer de tener como amigos: Gabri, Álvaro, Rosa, María, Almu, Taxen, Diego, Martin, María, Laura, Nacho, Marta, Santi, Fer, Álvaro, Álvaro, David… Y también a más de un profesor debo mencionar por su paciencia, porque ha sido importante en mi vida y me ha caído genial :) me refiero a David, Irene, Soledad, María Isabel, Gloria… Y a los que no os he puesto, tranquilos, que nunca me olvido de vosotros :D

Venga va, también le agradezco a Zorbas (el gatito de mi novia) el pasearse por encima de mi portátil mientras yo escribía estas líneas y "escribir" cosas sin sentido. Y en general, a todo el mundo que ha formado y forma parte de mi vida, gracias. Chicos, chicas, animales… POR FIN HE TERMINADO!!

RESUMEN

El nacimiento de la telefonía móvil hace varias décadas, generó un mercado que ha ido creciendo día a día, haciendo que se invierta cada vez más tiempo y dinero en la tecnología unida a este concepto. Actualmente, no se usan teléfonos móviles simples, sino que se han convertido en *smartphones*: verdaderas obras de ingeniería capaces de incluir sensores de movimiento o posición, conectividades Wi-Fi o *bluetooth*, cámaras de foto y vídeo, y todo un sinfín de posibilidades y herramientas.

Este desarrollo hizo que fuera necesaria la creación de sistemas operativos avanzados y potentes, capaces de dar uso a estas características. Para ello, desde ese tiempo hasta la actualidad, se están desarrollando todo tipo de aplicaciones para estos terminales, ya sean traductores, navegadores, videojuegos o herramientas de reconocimiento de actividades. Esta última es en la que se centra este Trabajo Final de Grado.

Este Trabajo Final de Grado consiste en la creación de una aplicación para el sistema operativo Android y cuyo objetivo es el reconocimiento de patrones de actividad del usuario, es decir, qué está haciendo el usuario en ese momento. La aplicación, gracias a los datos recibidos por el acelerómetro integrado en el terminal *smartphone* y a una red de neuronas artificiales que corre en su interior, detecta la actividad que está realizando el usuario (correr, caminar, estar sentado, subir escaleras…) y contabiliza el tiempo total que el usuario ha estado realizando dicha tarea. Para ello, y con el fin de captar al mayor número de usuarios posibles, se implementa una interfaz realmente sencilla e intuitiva para que, al instante, el usuario ya sepa manejar la aplicación.

Para la realización de este proyecto, se cuenta con el sistema operativo Android y las redes neuronales que ofrece la librería de Weka. La programación se realiza siguiendo la arquitectura Modelo-Vista-Controlador y la interfaz de la aplicación es sencilla e intuitiva. Las pruebas que se han realizado con los usuarios finales demuestran que tanto la aplicación como la red de neuronas que lleva integrada cumplen con los requisitos de fiabilidad, usabilidad, rendimiento, etc. Se ha realizado una aplicación que cumple con los objetivos establecidos tanto de precisión (con un porcentaje de acierto del 97% aproximadamente) como de rendimiento, reconociendo las actividades físicas del usuario en tiempo real.

De forma personal, este Trabajo Final de Grado me ha servido para familiarizarme con los sistemas operativos móviles, con redes neuronales artificiales, su uso en terminales móviles y me ha enseñado a afrontar la totalidad de un proyecto software, desde el primer análisis hasta la aceptación final del sistema, logrando recabar los conocimientos necesarios para ello.

TABLA DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Escenario del proyecto

El reconocimiento de actividades de un usuario pretende averiguar qué acción está realizando. Generalmente se utiliza para caminar, correr y montar en bicicleta, pero las posibilidades son mucho más amplias. Antes de la aparición de los terminales *smartphone*, había diferentes técnicas para determinar la actividad del usuario: El GPS y el acelerómetro.

A través de una serie de mediciones del GPS y el tiempo empleado para recorrer una distancia, se podía determinar la velocidad del usuario y, por ende, predecir la actividad en función de dicha velocidad. Aunque esta técnica no es muy fiable, sigue estando extendida en muchas aplicaciones.

Por otro lado está el acelerómetro, el cual permite determinar el movimiento de un objeto en las tres dimensiones. Combinado, al igual que el GPS, con el tiempo en la toma de datos, se calcula el flujo del movimiento del objeto y se determina la acción realizada por el usuario.

Para que el reconocimiento de actividades sea efectivo y eficiente, una de las opciones disponibles es programar una red de neuronas artificial. Eligiendo una serie de actividades, que luego podrá expandirse a otras acciones, se realiza un exhaustivo entrenamiento de la red para que aprenda y el error sea mínimo. Una vez finalizado el entrenamiento, la red de neuronas está lista para indicar cuál es la actividad que está realizando el usuario.

1.2. Objetivos

El principal enfoque que atañe a este Trabajo Final de Grado, se centra en el uso de redes neuronales en el sistema operativo Android. En concreto, se utilizará una red neuronal para el reconocimiento de patrones de actividad del usuario final. Para lograrlo, se deberán cumplir con los siguientes objetivos:

- Obtención de un conjunto de datos de entrenamiento lo más amplio y bueno posible, para que la red neuronal entrenada clasifique de forma correcta los datos que reciba en un futuro.
- Estudio de las redes neuronales programadas en Java y su tiempo de entrenamiento y ejecución para obtener el método más ligero y fiable que pueda ser utilizado en un terminal con sistema Android.
- Diseño y desarrollo de una red neuronal que, una vez entrenada con los datos de entrenamiento obtenidos, clasifique de forma correcta los datos de entrada recibidos.
- Estudio y aprendizaje de la programación para Android, incluyendo sus limitaciones a nivel de hardware y los sensores de los que se debe hacer uso, en este caso, el acelerómetro.
- Diseño y desarrollo de una aplicación que obtenga datos del acelerómetro del teléfono y, mediante el uso de la red de neuronas, indique la actividad que está realizando el usuario.
- Realización de pruebas unitarias de todos los apartados y completas de la aplicación, analizando los resultados obtenidos y realizando los ajustes necesarios.

- Documentación de todo el procedimiento asociado a este Trabajo Final de Grado, incluyendo el diseño y las pruebas de la aplicación final.

1.3. Gestión del proyecto

En la imagen adjunta en el Anexo II: Planificación (Ilustración 15) se puede ver la planificación real que se ha obtenido del Trabajo Final de Grado una vez finalizado el mismo.

Como puede verse, las tareas más duraderas han sido la documentación del proyecto en general, que ha abarcado de principio a fin todo el proceso de este trabajo, y la implementación de la aplicación. La documentación cumple con lo esperado debido a que cada fase del proyecto necesita ser documentada.

En cuanto a la aplicación, se entiende la duración de esta tarea, ya que uno de los objetivos era aprender a programar para el sistema operativo Android. Si bien la teoría es importante, donde realmente se aprende es en la práctica, y muchas de las pruebas que se han realizado han sido incorporadas a la aplicación final.

Por último, cabe indicar que se ha seguido un ciclo de vida en cascada, que consiste en ordenar las etapas del proceso de desarrollo de tal forma que el inicio de cada etapa se produce una vez finalizada la etapa anterior.

Se han realizado cada una de las tareas en el orden correcto, salvo la documentación, que abarca todo el proyecto. Aproximadamente una vez a la semana, se realizaba un seguimiento del estado actual del trabajo, evaluando posibles problemas existentes y mejoras futuras.

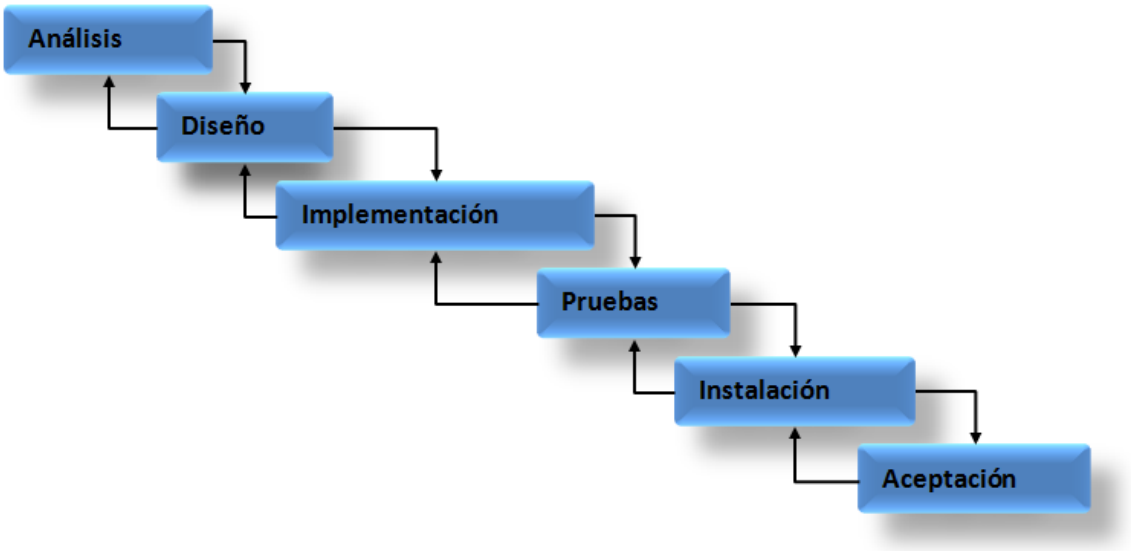


Ilustración 1. Modelo de desarrollo en cascada.

1.4. Estructura de la memoria

Después de una breve introducción, a continuación se detalla la estructura del resto del documento:

1. **INTRODUCCIÓN**: Se realizará un pequeño resumen del universo de la aplicación, estructura de la memoria, objetivos primarios…

2. **ESTADO DE LA CUESTIÓN**: Se analizarán las últimas novedades y recientes desarrollos en el mundo tecnológico actual que tengan algún tipo de relación con el uso de terminales móviles para el reconocimiento de patrones de actividad del usuario.

3. **OBJETIVOS**: Se enumerarán los objetivos de este Trabajo Final de Grado, tanto el objetivo general como los específicos.

4. **DISEÑO DE LA APLICACIÓN**: Se detallará el trabajo realizado ofreciendo una descripción de los requisitos, arquitectura y diseño de la aplicación.

5. **IMPLEMENTACIÓN DEL SISTEMA**: Se detallará más profundamente el desarrollo de la aplicación, sus componentes y sus funciones.

6. **PRUEBAS Y EVALUACIÓN**: Se mostrarán los resultados obtenidos en la fase experimental de este proyecto, demostrando que se cumple con lo exigido en el diseño.

7. **CONCLUSIONES**: Se narrarán las conclusiones obtenidas tras la realización de este proyecto.

8. **LÍNEAS FUTURAS**: Figurarán los trabajos futuros que se pueden llevar a cabo sobre la base del trabajo realizado.

9. **GLOSARIO**: En este capítulo se incluirán todos aquellos términos, siglas y demás que necesiten de una pequeña explicación.

10. **REFERENCIAS**: Se detallan cada una de las referencias utilizadas para este documento con toda la información disponible sobre las mismas.

11. **ANEXO I: MANUAL DE USUARIO**: Aquí se explican los requisitos de la aplicación para hacerla funcionar, cómo instalar dicha aplicación y como ponerla en funcionamiento.

12. **ANEXO II: PLANIFICACIÓN**: Se detalla la planificación inicial del proyecto que se realizó en su momento comparada con la planificación real al final del mismo.

13. **ANEXO III: PRESUPUESTO**: Aquí se desglosa el dinero invertido en este Trabajo Final de Grado y una pequeña explicación del resultado de la inversión.

2. ESTADO DE LA CUESTIÓN

2.1. Android

Android (1) es un sistema operativo basado en el núcleo principal de Linux, desarrollado principalmente para su uso en *smartphones* y *tablets*. Actualmente es el sistema operativo con más cuota del mercado, llegando al sobrepasar el 50% a finales de 2011 (2) como puede verse en la Ilustración 2.

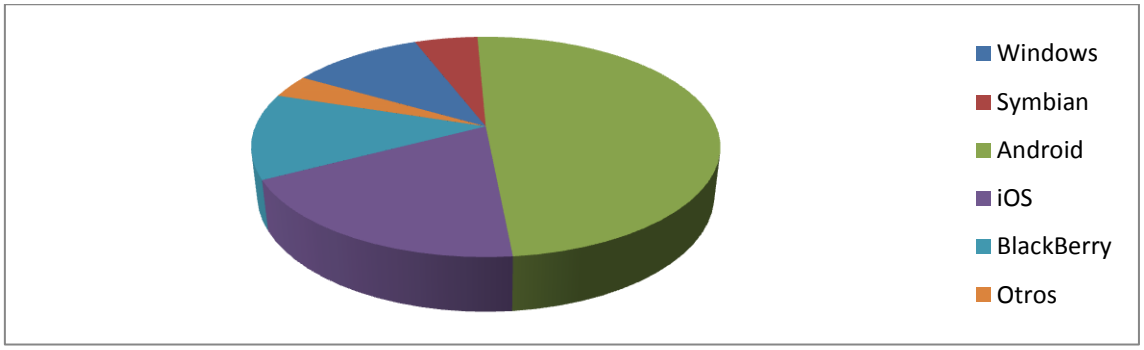


Ilustración 2. Cuota de mercado de telefonía móvil a finales de 2011.

2.1.1. Historia

Android fue desarrollado inicialmente por la empresa Android Inc., la cual pasó a ser propiedad de Google a mediados de 2005. En aquel entonces, y con la colaboración de varios cofundadores de Android, se especuló sobre si Google planeaba entrar al mercado móvil.

La Open Handset Alliance es una alianza de 84 compañías que se dedica al desarrollo de estándares para telefonía móvil. Liderada por Google, fue fundada en septiembre de 2007, y su primer lanzamiento fue el sistema Android basado en el kernel 2.6 de Linux.

Su primera versión fue publicada el 23 de septiembre de 2008, y desde entonces no se ha parado de actualizar a este gigante de la telefonía móvil. Entre las versiones más conocidas y utilizadas están la 2.3 (también conocida como Gingerbread) lanzada en diciembre de 2010 y siendo la más extendida en terminales *smartphones*, la versión 3.0 (Honeycomb) que fue adaptada para su uso en *tablets*, y la versión 4.0 (Ice Cream Sandwich), disponible desde diciembre de 2011, cuyo principal objetivo es unificar todos los sistemas operativos de *smartphones*, *tablets* y cualquier dispositivo que utilice Android.

2.1.2. Características principales

Entre las múltiples características que brinda este sistema operativo, se pueden destacar las siguientes:

- **Diseño del dispositivo**: El sistema ha sido adaptado para cualquier tamaño de pantalla y pantallas VGA, así como gráficos en dos y tres dimensiones siguiendo las especificaciones OpenGL 2.0. También se mantiene el diseño de los teléfonos tradicionales. Se ha buscado la mayor facilidad de uso posible sin perder la elegancia y la rapidez del sistema.

- **Base de datos**: Se utiliza como motor de base de datos SQLite, un sistema muy sencillo y liviano que ofrece todo lo necesario para el almacenamiento y utiliza poca cantidad de recursos, algo muy necesario en terminales móviles.

- **Conectividad**: Android soporta las tecnologías GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE y WiMAX para la conectividad del terminal móvil.

- **Mensajería**: Aparte de los SMS y mensajes multimedia, incluyendo mensajería de texto, Android ha incluido el servicio C2DM como otro servicio de mensajería ofrecido.

- **Navegador web**: El navegador disponible en el sistema operativo Android está basado en el motor WebKit de renderizado, el cual es de código abierto. Unido al motor JavaScript V8 perteneciente a Google Chrome, ha obtenido una puntuación del 93% en los test Acid3.

- **Soporte Java**: A pesar de que la mayoría de aplicaciones están desarrolladas en Java y se utiliza este lenguaje, no existe una máquina virtual para Java en Android. Antes de ser ejecutado en los terminales, el código se compila en un ejecutable tipo Dalvik y se ejecuta en una máquina virtual. Las máquinas Dalvik fueron diseñadas específicamente para Android y optimizadas para terminales con recursos limitados, como un *smartphone* o una *tablet*. Aunque no tiene soporte oficial para J2ME, se puede agregar mediante aplicaciones existentes.

- **Multimedia**: En lo referente a formatos de audio, vídeo e imágenes, Android soporta los siguientes formatos: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

- **Hardware adicional**: Dado que es un sistema operativo principalmente para *smartphones*, soporta el hardware que éstos traen consigo, como las cámaras de foto y vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, termómetro, aceleración 2D y 3D.

- **Entornos de desarrollo**: Se ha desarrollado un emulador de dispositivos que dispone de cualquiera de las versiones existentes de Android para probar las aplicaciones que hagan los desarrolladores. También se proporcionan herramientas de depuración y de análisis de rendimientos. Se utiliza Eclipse (a partir de la versión 3.4) como entorno de desarrollo en conjunto con el *plug-in* ADT (Herramientas para el Desarrollo de Android).

- **Google Play** (3): Catálogo de aplicaciones (la mayoría de ellas gratuitas) soportado por Google y que permite la descarga e instalación de multitud de aplicaciones en dispositivos Android sin necesidad de un ordenador.

- **Multitáctil**: Android dispone de forma oficial del soporte para pantallas multitáctiles, las cuales aparecieron con el HTC Hero. Aunque al principio se desactivó esta característica a nivel del núcleo, más tarde Google publicó unas actualizaciones para activar de nuevo este soporte de forma nativa que muchos consideran necesario.

- **Bluetooth**: Desde la versión 1.5 se dispone del soporte para A2DF y AVRCP. Posteriormente, desde la segunda versión, se incluyeron las opciones de transferencia de ficheros y exploración del directorio telefónico. Por último, el marcado de voz y la transferencia de contactos llegó con la versión 2.2.

- **Videollamada**: Mediante Google Talk, que es una aplicación para conversaciones en tiempo real, se pueden realizar videollamadas, todo ello desde la versión 3.0, conocida como Honeycomb.

- **Multitarea**: Aunque algunos simplemente emulan esta característica, Android la ofrece de forma real. Traducido al rendimiento, permite que aquellas aplicaciones que no estén siendo utilizadas pero estén en funcionamiento reciban ciclos de reloj.

- **Voz**: La búsqueda en Google a través de voz está disponible como "Entrada de Búsqueda" desde la versión inicial del sistema.

- **Tethering**: Se conoce al *tethering* como el uso del terminal como punto de acceso a internet, ya sea alámbrico o inalámbrico. Se incorporó oficialmente en la versión 2.2, aunque estaba disponible desde la 1.6 con aplicaciones de terceros. Esto permite a un ordenador conectarse a internet mediante la conexión 3G del móvil.

2.1.3. Arquitectura

La arquitectura del sistema operativo Android se diferencia en 5 grandes bloques, cada uno con sus utilidades y sus características. En la Ilustración 3 puede verse un esquema de los mismos:

- **Núcleo de Linux**. Es el componente base de todo el sistema operativo. Contiene todos los controladores necesarios para el hardware del terminal así como la seguridad, gestión de memoria, procesos y aplicaciones, y la pila de red.

- **Librerías**. Todas las librerías de Android están desarrolladas en el lenguaje C/C++ y proveen a los desarrolladores de aplicaciones (mediante el *framework*) todas las características del sistema operativo. Por ejemplo, las bases de datos del sistema, librerías de gráficos 2D y 3D, etc.

- **Android Runtime**. Es un conjunto básico de bibliotecas que proveen la mayoría de las funciones del lenguaje Java. Toda aplicación Android se ejecuta en máquinas virtuales independientes, llamadas máquinas virtuales Dalvik. Estas virtualizaciones están optimizadas para sistemas con recursos limitados (como un teléfono móvil), están basadas en registros y contienen clases de Java transformadas al formato Dalvik.

- **Framework de aplicaciones**. Es un conjunto de APIs del sistema operativo Android que actúan de forma directa con las aplicaciones desarrolladas. Los desarrolladores tienen un acceso completo a esta interfaz de programación y se diseñó para simplificar la reutilización de los componentes. Siempre que la seguridad impuesta lo permita, una aplicación puede publicar sus capacidades y luego otra hacer uso de las mismas.

- **Aplicaciones**. Es la capa superior del sistema y contiene todos los programas que puede utilizar el usuario, desde la lista de contactos del teléfono, navegador web, hasta los juegos de los que disponga. Toda aplicación está basada en el lenguaje Java que, posteriormente, se convierte a formato Dalvik para su ejecución en los terminales. Existen multitud de aplicaciones en la tienda oficial de Android, anteriormente conocida como Android Market y ahora Google Play.

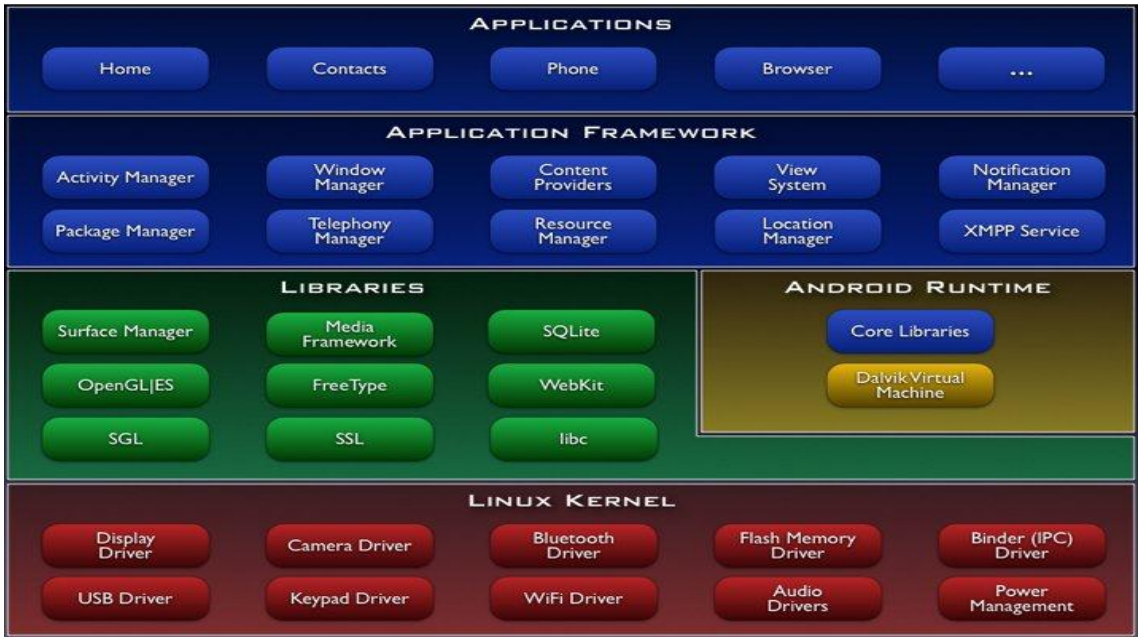


Ilustración 3. Arquitectura del sistema operativo Android

2.1.4. Versiones

Android no para de crecer. Desde la primera versión, que salió al mercado en septiembre de 2008, ya se han generado 8 versiones propiamente dichas. A un ritmo medio de una versión cada 6 meses, actualmente la más novedosa es la 4.0. Como curiosidad añadida, Google utiliza nombres de postres (por supuesto en inglés) ordenados alfabéticamente para sus versiones.

- **1.0 – Android**. Fue lanzada al mercado el 23 de septiembre de 2008 a través del terminal HTC Dream y contaba con algunas características que, a día de hoy, parecerían ínfimas: conectividad Wi-Fi y Bluetooth, Android Market (actualmente conocido como Google Play), navegador web, cámara, correo electrónico, servicios de Google Mail, Contacts, Calendar, Maps (con Street View incluido), Sync, Search y Talk, mensajería instantánea, Media Player, notificaciones y YouTube.

- **1.1**. Liberada el 9 de febrero de 2009, su principal cometido fue arreglar algunos problemas existentes en su predecesora, mejorar algunas características e incluir nuevas funcionalidades. Aunque era conocida como Petit Four, este nombre no fue utilizado oficialmente.

- **1.5 – Cupcake**. Vio la luz el 30 de abril de 2009, apenas 2 meses después que la versión 1.1. Entre sus principales novedades y actualizaciones cabe destacar la posibilidad de subir vídeos a YouTube e imágenes en Picasa, un nuevo modelo de teclado con predicción de textos, nuevos iconos y *widgets* de escritorio, transición animada entre las diferentes pantallas de inicio y el soporte para A2DP y AVRCP de Bluetooth.

- **1.6 – Donut**. 5 meses después de Cupcake, la versión 1.6 de Android fue instalada en algunos terminales. Cuenta con búsquedas por voz mejoradas, actualizaciones para conectividad CDMA/EVDO, 802.1X y VPN, resolución de pantallas WVGA, un *framework* de gestos y navegación gratuita *turn-by-turn* de Google.

- **2.0 y 2.1 – Eclair**. Con tan solo un mes de diferencia, el 26 de octubre de 2009 salió al público la nueva versión 2.0. Incluía mejoras en la velocidad del hardware, más tamaños de pantalla y resoluciones, soporte para HTML5, mejoras en Google Maps, flash para la cámara de fotos, zoom digital, teclado virtual mejorado y una mejor relación de contraste para los fondos.

- **2.2 – Froyo**. Se lanzó el 20 de mayo de 2010 incluyendo una mejora general del sistema en cuanto a memoria y rendimiento, mayor velocidad en la ejecución de aplicaciones gracias al uso de JIT, integración del motor JavaScript V8, soporte mejorado de Microsoft Exchange, funcionalidad nativa de *tethering* por USB y Wi-Fi, desactivación opcional de la red de datos, cambios rápidos de idiomas del teclado, marcación por voz, Adobe Flash 10.1 y soporte para pantallas de alta resolución como 4'' 720p.

- **2.3 – Gingerbread**. Liberada el 6 de diciembre de 2010, ha logrado la supremacía en cuota de mercado frente a las otras versiones de Android. Sus cambios incluyen soporte a dispositivos móviles, actualización del diseño, soporte a pantallas extra grandes, telefonía VoIP SIP, efectos de audio nuevos, tecnología NFC, funcionalidad de cortar, copiar y pegar a lo largo del sistema, teclado multitáctil, mejoras en el desarrollo de juegos, soporte nativo para sensores como el giroscopio, administración mejorada de la energía, y se modificó el sistema de ficheros a *ext4*.

- **3.X – Honeycomb**. Viendo la luz el 22 de febrero de 2011, la versión Honeycomb incluye un soporte para *tablets*, escritorios con gráficos 3D, mejoras en el sistema multitarea y el navegador web, soporte para videollamadas con GTalk, mejor soporte para redes Wi-Fi, soporte para periféricos conectados con USB, como teclados, ratones, HUBs o dispositivos de juegos y un soporte opcional para redimensionar el tamaño de las aplicaciones que se desarrollaron para móviles y se utilizaban en *tablets*, con pantallas mayores.

- **4.0 – Ice Cream Sandiwch**. Es la más novedosa y joven de las versiones de Android, y su principal meta es la unificación de todos los sistemas Android en una misma versión. Lanzada a finales de 2011, incluye una interfaz mejorada, aceleración por hardware de los gráficos, multitarea mejorada, gestor de tráfico de datos, corrector de textos mejorado y rediseñado, capturas de pantalla, nuevas utilidades para la cámara de fotos y vídeo, reconocimiento de voz y facial, unificación del *framework* de desarrollo para aplicaciones, herramientas de control de aplicaciones, soporte para el formato de vídeo MKV y soporte nativo para lápices táctiles Stylus.

A continuación, en la Ilustración 4, se muestra un gráfico en el que se puede apreciar claramente la superioridad de cuota de la versión 2.3.3. Aunque la versión 4.0 y posteriores son aún muy jóvenes, poco a poco pasarán a copar la mayoría absoluta en los terminales con Android ya que se diseñaron para unificar todos los dispositivos que utilizaban este sistema operativo.

Plataforma	%
4.0.x *Ice Cream Sandwich*	1,6%
3.x.x *Honeycomb*	3,3%
2.3.x *Gingerbread*	62,0%
2.2 *Froyo*	25,3%
2.1 *Eclair*	6,6%
1.6 *Donut*	0,8%
1.5 *Cupcake*	0,4%

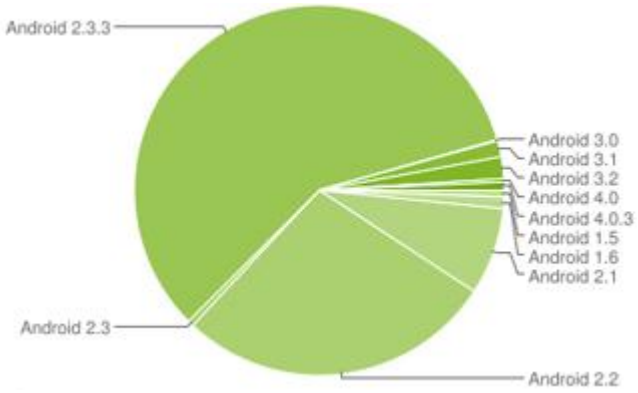


Ilustración 4. Cuota de mercado de las versiones de Android

2.1.5. Comparativa con otros sistemas operativos

Una de las características principales de este proyecto es su diseño para ser ejecutado en el sistema operativo Android, concretamente su versión Gingerbread 2.3. Se ha detallado en este documento todo lo referente a sus características, pero ahora queda aclarar la elección de este sistema y no otro.

Aunque la variedad de sistemas operativos para *smartphones* es bastante amplia, en esta comparativa se revisarán los sistemas operativos con mayor cuota de mercado. La empresa Gartner (2) ha realizado un estudio predictivo para ver la cuota de mercado de los sistemas operativos móviles en 2015 (ver Tabla 1), con aplastante resultado de Android.

Sistema Operativo	2010	2011	2012	2015
Android	22,70%	38,50%	49,20%	48,80%
iOS	15,70%	19,40%	18,90%	17,20%
Windows Phone	4,20%	5,60%	10,80%	19,50%
BlackBerry	16,00%	13,40%	12,60%	11,10%
Symbian	37,60%	19,20%	5,20%	0,10%

Tabla 1. Cuota de mercado para los sistemas operativos móviles.

Como puede verse, Android es la apuesta segura con la mitad de terminales móviles del planeta, y su constante crecimiento de cuota indica que seguirá así bastante tiempo. En esta comparativa se enfrentarán a los tres sistemas con mayor cuota de mercado prevista: Android, iOS y Windows Phone.

En lo referente al **hardware del teléfono**, Android está muy extendido en diversos terminales diferentes y soporta múltiples arquitecturas y dispositivos hardware. iOS, aunque su soporte es muy bueno, no está muy extendido en diversidad de terminales móviles. Windows Phone está entre ambos, y ofrece menos potencia de hardware pero con algo más de variedad.

Para el **desarrollo de aplicaciones**, Android es un sistema de código abierto que puede ser desarrollado en múltiples equipos. Por otro lado, tanto iOS como Windows Phone son de código cerrado. Además, Android ofrece a sus desarrolladores un SDK completo, libre, y disponible para cualquier sistema operativo de PC.

Hablando de la **memoria externa** del terminal, cada sistema ha optado por un método diferente. iOS no permite la inserción de tarjetas de memoria, utilizando íntegramente la

memoria interna del teléfono. Windows Phone permite el uso de una, y tan solo una tarjeta, que luego no podrá ser retirada. En realidad, es equiparable a convertir la memoria externa en interna. Por último, Android, permite el uso de una memoria externa así como dicha memoria a través de USB.

En cuanto a la **velocidad de ejecución**, Android es el único que permite una ejecución multitarea, es decir, de varios procesos al mismo tiempo (siempre que el hardware lo permita). Por otro lado, aunque iOS y Windows Phone dispongan de varios núcleos de ejecución, su sistema no es multitarea y no aprovecha toda la capacidad del *smartphone*.

Para finalizar, la **interoperabilidad** es un aspecto a tener en cuenta. Todos los terminales, independientemente del hardware, pueden soportar una aplicación destinada a su sistema operativo. En este aspecto no se puede declarar a un vencedor ya que todos cumplen las expectativas.

En resumen, Android es, sin lugar a dudas, el mejor sistema operativo para el desarrollo de aplicaciones. No solo por su creciente cuota de mercado, sino porque ofrece mayores posibilidades y ventajas tanto a desarrolladores como a usuarios finales.

2.2. Acelerómetro

El acelerómetro es un dispositivo que se encarga de obtener mediciones de aceleración y fuerzas inducidas por la gravedad en los tres ejes. En otras palabras, y de forma mucho más simple, es un sensor que detecta el movimiento y el giro. Se utiliza principalmente para detectar la posición del terminal y los movimientos que se realizan sobre éste.

2.2.1. Historia

El primer acelerómetro que se concibió fue el conocido como acelerómetro mecánico (4). Este dispositivo se valía de la segunda ley de Newton, por la cual la fuerza equivale a masa por aceleración. Se construyó uniendo un objeto de cierta masa con un dinamómetro orientado en la dirección de la cual se deseaba medir la aceleración. El dinamómetro permitía calcular la fuerza ejercida sobre este. Gracias a ello y a la masa del objeto, era posible calcular la aceleración producida.

Más tarde se creó el acelerómetro piezoeléctrico, el cual se valía de que, al comprimir un retículo cristalino piezoeléctrico, este producía una carga eléctrica directamente proporcional a la fuerza aplicada en la compresión. Estas cargas se capturaban y medían en un osciloscopio o voltímetro.

Existen otros tipos de acelerómetro como el acelerómetro de efecto Hall, el cual utiliza una masa sísmica con un imán y un sensor de efecto Hall que detecta los cambios del campo magnético. También existen los acelerómetros de condensador, el cual mide el cambio de la capacidad eléctrica de un condensador. Dentro se coloca una masa sísmica entre las placas de dicho condensador que, al desplazarse, provoca un cambio en la corriente.

Inicialmente todo acelerómetro estaba destinado a un solo eje, por lo que el acelerómetro debía colocarse paralelo a él. En la actualidad, y gracias a las nuevas tecnologías, existen acelerómetros de tres ejes (X, Y, Z) construidos en un simple chip que ya incluye la parte de procesado de señales.

La mayoría actualmente se basa en la tecnología MEMS y el traspaso térmico (5). La aceleración que se provoca sobre el sensor genera cierta temperatura y ésta se puede medir de forma fiable. Esta técnica ofrece muchas ventajas respecto a los antiguos acelerómetros que requerían una estructura sólida. Entre las principales ventajas están la desaparición de una estructura sólida (y en general las estructuras mecánicas internas), lo que aligeraba significativamente el peso del acelerómetro, y la precisión del mismo, que se considera muy buena.

2.2.2. Funcionamiento

Los acelerómetros más utilizados son los basados en el cambio térmico y los que utilizan condensadores. Los primeros se valen de la aceleración provocada al sensor, que genera una cierta temperatura, la cual va relacionada con la fuerza de la gravedad. Los que utilizan condensadores, contienen una pequeña masa dentro del condensador que, al moverse (ya sea por el movimiento del teléfono o por la gravedad), provoca un cambio en la capacidad del condensador.

2.2.3. Uso en smartphones

Su uso en los s*martphones* está muy extendido, debido al amplio abanico de posibilidades que ofrece. La mayoría de ellos lo utilizan para rotar la pantalla en función de si se encuentra vertical u horizontal, y también para detectar la toma de una fotografía y rotar la foto automáticamente.

Otro de los grandes usos que se da a este sensor proviene de terceros que desarrollan sus propias aplicaciones. Por ejemplo, en multitud de videojuegos de equilibrio o control de vehículos se utiliza esta tecnología. Pero algunos van más allá y han creado, entre otras, aplicaciones para el control remoto de un coche de juguete, el cual se maneja mediante el acelerómetro, o aplicaciones que permiten controlar la reproducción de música y la lectura de mensajes con el simple hecho de agitar el terminal.

2.3. Redes de neuronas artificiales

Una red de neuronas artificial es un paradigma de aprendizaje que consiste en diseñar un programa capaz de aprender y saber cuál es el resultado que debe dar en función de los datos de entrada. La idea es simular las redes neuronales de los animales a través de modelos matemáticos y se asemeja a un sistema de interconexión de nodos que colabora para producir un resultado (6).

Como todo el mundo sabe, el cerebro humano y los ordenadores son muy diferentes en lo que se refiere a la computación de datos. Si bien una persona es capaz de aprender y reconocer patrones, esta acción no es nada sencilla para una máquina. Por el contrario, un ordenador es capaz de realizar operaciones mecánicas y matemáticas de una forma asombrosamente rápida, al contrario que una persona, dependiendo de la complejidad del problema.

Las primeras redes neuronales conocidas datan de 1959 de la mano de Frank Rosenblatt gracias a estudios anteriores de McCulloch y Pitts, que demostró la habilidad de varias neuronas conectadas entre sí para realizar funciones lógicas lineales. Dichas redes neuronales, que poseían capacidad de aprendizaje, se conocen hoy día como perceptrón simple (7).

2.3.1. Perceptrón simple

El perceptrón simple es el modelo más sencillo de una red neuronal. Se compone de un grupo de neuronas de entrada, las cuales reciben los datos del exterior, y una neurona de salida, que devuelve el resultado de la clasificación. Entre las principales ventajas de este modelo se encuentra su capacidad para aprender a clasificar los datos, pero cuenta con un gran inconveniente, y es que tan solo puede utilizarse en entornos de clasificación lineales como demuestra la Ilustración 5 (6).

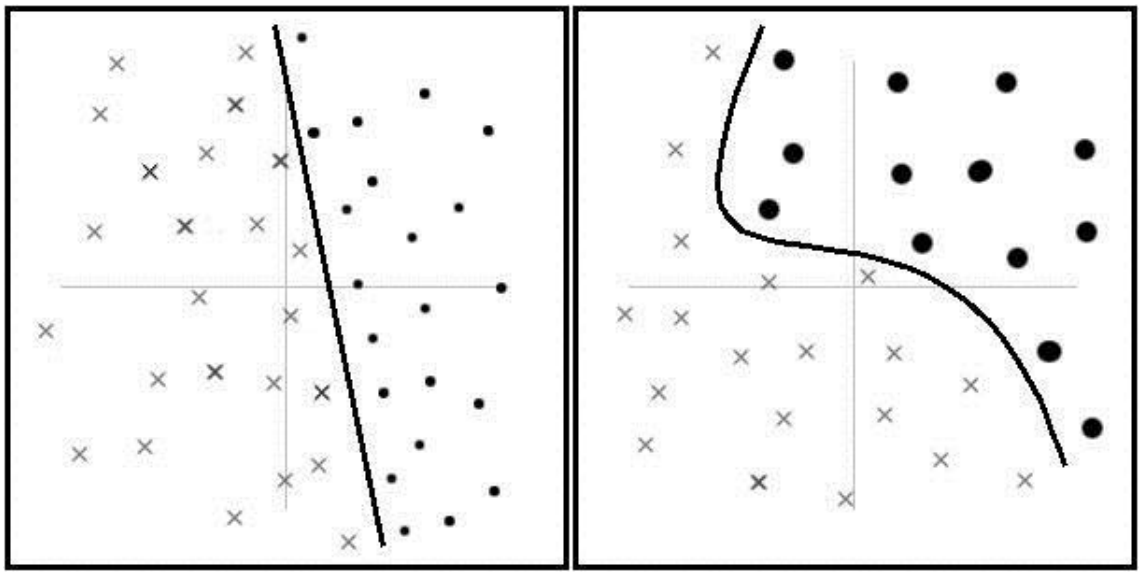


Ilustración 5. Clasificación lineal (izquierda) y no lineal (derecha) de los datos.

Como se ha indicado en el párrafo anterior, este modelo presentaba algunas limitaciones como el uso exclusivo en entornos lineales y podían ser solventadas mediante la combinación de varios perceptrones simple. Esta idea surgió en 1986 y trajo consigo el modelo conocido como perceptrón multicapa.

2.3.2. Perceptrón multicapa

Se descubrió que la combinación de varios perceptrones simples podía resolver los problemas no lineales y se dio lugar al perceptrón multicapa (8), ideado para la resolución de problemas que no se pueden separar linealmente. Se basa en el uso de múltiples capas intermedias entre la entrada de datos y la salida producida (Ilustración 6). Dichas capas están compuestas de varios nodos, cada uno conectado con todos los nodos de la siguiente capa, para colaborar en la elaboración de una respuesta.

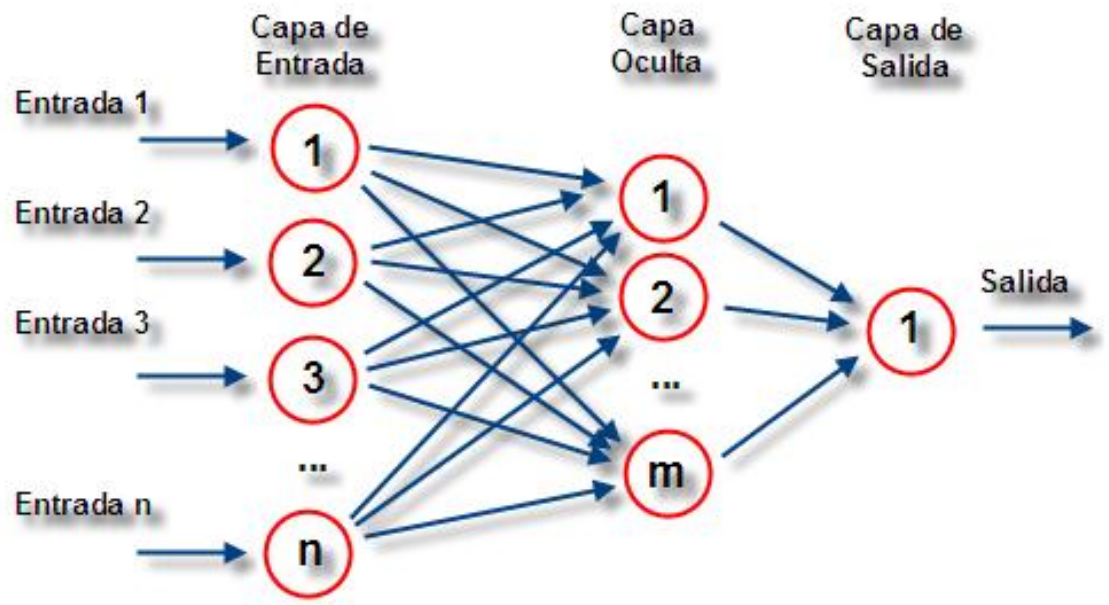


Ilustración 6. Ejemplo de perceptrón multicapa.

Cada uno de los nodos de las capas intermedias se compone de un vector de pesos, recibe un vector de entradas y produce un vector de salidas. Los pesos se utilizan para realizar el cálculo de la salida, e indican cuánto se debe tener en cuenta los datos de entrada asociados a ese peso. Para ello, se calcula el producto de una entrada por su peso y se envía a la salida.

Para lograr que los pesos de las neuronas sean los apropiados, estas son entrenadas mediante diferentes técnicas. Una de las técnicas más conocidas es la retro-propagación o *backpropagation*. A continuación se detallará qué es el aprendizaje automático y como se utiliza en las redes neuronales.

2.3.3. Aprendizaje automático

El aprendizaje automático de una red de neuronas se utiliza para entrenarla y que produzca una salida lo más ajustada a la realidad. Para ello, se provee a la red neuronal de una serie de datos de entrada y de la salida correspondiente. En este caso, la red neuronal utiliza los datos y la previsión de la salida para calcular los pesos óptimos de cada nodo o neurona. Gracias a un buen entrenamiento, la red de neuronas puede clasificar los datos de entrada recibidos con una precisión cada vez mayor.

Existen varios algoritmos de entrenamiento según el tipo de red neuronal. En los perceptrones multicapa, que son redes con conexión hacia adelante, se utiliza el algoritmo de *backpropagation* (9).

2.3.3.1. Retro-propagación

Este algoritmo es el más utilizado por los perceptrones multicapa. Se basa en minimizar el error generado por la red neuronal con respecto al resultado. Para ello, se crea una red neuronal con una serie de pesos aleatorios. Se le pasa a la red neuronal una serie de datos de entrada para que esta calcule la salida correspondiente. Después, se compara la salida generada por la red con la salida que debía haber generado (conocida previamente). Ese error se devuelve hacia atrás desde la capa de salida, y cada neurona de la capa oculta recibe la

corrección que debe realizar proporcional al error generado. Cada neurona reajusta su peso para solventar el error cometido y se procede con el siguiente grupo de datos de entrada.

2.3.3.2. Validación del entrenamiento

Una vez entrenada la red neuronal, se debe proceder a la validación de dicho entrenamiento. El método sencillo de validación es volver a introducir los datos de entrenamiento en la red neuronal y comprobar el porcentaje de acierto que se obtiene. Pero este sistema trae un problema, y es que la red puede haberse adaptado a los datos de entrenamiento y no reflejar la realidad.

Para evitar este contratiempo se utiliza la técnica de validación cruzada (10). Esta técnica consiste en dividir los datos de los que se dispone en varios subconjuntos. Uno de esos conjuntos se utiliza como datos de prueba o validación, mientras que el resto se utilizan como entrenamiento (Ilustración 7). Cada una de estas pruebas devuelve un porcentaje de éxito, y la validación final se calcula mediante la media de dichos porcentajes.

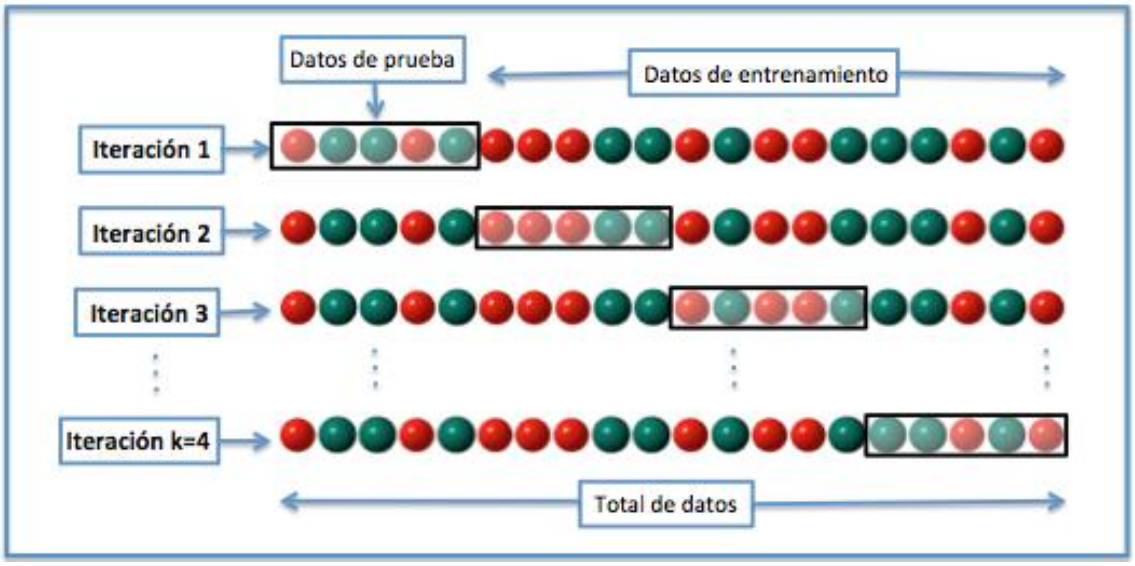


Ilustración 7. Validación cruzada de 4 subconjuntos.

2.3.4. Uso en smartphones

El uso de este tipo de tecnología en terminales *smartphone* no está muy extendido, y se debe principalmente a que la ejecución de redes neuronales suele ser un trabajo que trae consigo una importante carga de proceso. Los *smartphone* tienen un hardware limitado y no pueden competir contra ordenadores corrientes en temas de procesamiento, pero aun así, se utiliza.

Un buen y muy extendido uso de esta tecnología es el reconocimiento de patrones de voz. Para ello, el teléfono recoge una muestra del habla de una persona e intenta determinar las palabras que ha dicho. Su entrenamiento se basa en muestras de frases de varias personas de todo tipo para lograr un buen entrenamiento y una tasa de error mínima. En dispositivos Android se utiliza para el buscador integrado del sistema, mediante el cual se puede realizar una búsqueda dictando la palabra deseada.

2.4. Aplicaciones para el reconocimiento de actividades

A continuación se detallarán una serie de aplicaciones que realizan comportamientos y funciones similares a la que se desarrollará y se explicará qué carencias tiene dicha aplicación y cómo se solventa en la nuestra.

2.4.1. Android

- **Sports Tracker Pro** (11): Es una aplicación bastante completa que monitoriza las actividades realizadas por el usuario, tales como correr, bicicleta, caminar, esquí, snowboard, vela y cinta de correr. La aplicación se basa en la información GPS obtenida, el tiempo de ejecución y la tarea elegida por el usuario. Además, muestra gráficos y estadísticas gracias al historial de acciones. Requiere la versión 1.5 de Android como mínimo. La principal carencia es que el usuario debe elegir manualmente la actividad que está realizando, mientras que nuestra aplicación lo hace de forma automática. Su precio es de 3,49 €.

- **Runastic Pro** (12): Al igual que la aplicación anterior, se basa en la información del GPS, el tiempo de ejecución del programa y los datos que introduce manualmente el usuario, tal como la actividad que realiza o los entrenamientos que desee cargar. Requiere Android 2.1 o superior. Tiene la misma carencia, aunque incluye un reproductor de música. Su precio es de 3,99 €.

- **All-in Podómetro +** (13): Es un podómetro muy completo, es decir, calcula el número de pasos, la velocidad del usuario y, de forma indirecta, puede predecir la distancia recorrida. No se basa en los datos de posicionamiento del GPS, sino que los podómetros utilizan un acelerómetro para calcular los pasos de usuario. Dispone de varias gráficas y estadísticas, y está calibrado a través de unas mediciones estándar. La carencia esencial de la aplicación es que no utiliza el GPS para mejorar los datos que calcula gracias al acelerómetro. Requiere Android 2.1 o superior y tiene un precio de 0,76 €.

2.4.2. Otros sistemas operativos

- **Podómetro PRO GPS +** (14): Esta aplicación desarrollada para los terminales de Apple se basa en las lecturas del acelerómetro y el GPS para determinar la actividad que realiza el usuario. Está disponible en varios idiomas (español, inglés, francés, alemán, italiano, portugués, y ruso). Requiere una versión de iOS mínima de 3.0 y es compatible con iPhone, iPod e iPad. Sin embargo, solo está disponible para este sistema operativo y no existe una aplicación de la misma compañía para Android o Windows Phone. Su precio es de 2,39 €.

- **SmartRunner** (15): Es una aplicación destinada al sistema operativo Windows Phone. Se vale del sistema GPS y el tiempo para los cálculos de distancia y demás estadísticas. Incluye listas de amigos y un servidor en el que guardar los datos de cada usuario. Es gratuita y requiere una versión mínima de Windows Phone 7.

En resumen, cada una de las aplicaciones utiliza el GPS, el acelerómetro, o ambas como es el caso de Podómetro PRO GPS +, pero nuestra aplicación ambas tecnologías además de estar destinada al sistema operativo con más cuota del mercado. Gracias a la red de neuronas artificiales previamente entrenada y calibrada, se determina, con los datos del acelerómetro, la actividad que realiza el usuario con tan solo encender la aplicación.

2.5. Herramientas empleadas

En este apartado se detallan las diferentes herramientas empleadas en la elaboración de este trabajo así como la aplicación asociada.

2.5.1. Android SDK

Android SDK es el kit de desarrollo de software necesario para la creación de aplicaciones que trabajan sobre el sistema operativo Android. Su descarga es gratuita desde la página oficial de desarrolladores de Android. Está disponible para los sistemas operativos Windows XP o superior, Mac OS X 10.5.8 o superior y Linux (requiere librería GNU C v2.7 o superior). Entre los principales componentes de este SDK podemos encontrar:

- **Herramientas SDK**: Es el componente principal del kit de desarrollo. Incluye todas las herramientas necesarias para el desarrollo y la depuración de las aplicaciones Android. Además permite la ejecución de un *smartphone* Android emulado para probar las aplicaciones.

- **Plataforma SDK**: La plataforma SDK es el sistema operativo sobre el que se probarán las diferentes aplicaciones. Contiene una versión adaptada del sistema operativo Android, en cualquier versión, para que el emulador pueda funcionar correctamente. Se requiere al menos la descarga de una versión de la plataforma.

- **Driver USB**: Desarrollado por Google, permite a los desarrolladores utilizar su propio *smartphone* para las pruebas de aplicaciones, en vez de utilizar un emulador. Para ello se conecta el *smartphone* a través del puerto USB del equipo. Es una técnica muy popular debido a que los terminales son más rápidos que el propio emulador.

- **Complementos**: Los complementos del kit de desarrollo permiten personalizar el entorno y el emulador según las necesidades de cada usuario. Debido a que cada versión de Android es específica para ciertos terminales, cada versión trae consigo unos complementos diferentes.

- **Ejemplos**: Los ejemplos contienen el código de aplicaciones que muestran cómo utilizar las características de Android. Así, un usuario puede consultar el código que necesita en busca de ayuda o documentación.

- **Documentación**: La documentación incluye la última versión del API de desarrollo para el sistema operativo Android en la versión elegida.

Una de las principales ventajas de este kit de desarrollo es su integración con entornos de desarrollo que facilitan enormemente el trabajo del desarrollador. El principal entorno utilizado para el desarrollo de aplicaciones Android es Eclipse IDE a partir de su versión 3.6, el cual se explica a continuación.

2.5.2. Eclipse IDE

Eclipse (16) es un IDE (entorno de desarrollo integrado) creado y diseñado por Eclipse Foundation destinado al desarrollo de aplicaciones. Se caracteriza por ser multiplataforma, lo que permite utilizarlo en diversos sistemas operativos, y soportar una importante cantidad de lenguajes de programación.

Aunque se le considera un IDE, no es solo eso. Más bien se podría considerar un marco de trabajo escalable a través de módulos (generalmente de terceros) que permiten la integración de herramientas, características y opciones para el desarrollo de nuevas aplicaciones. Uno de los módulos utilizados en este proyecto es el *plug-in* ADT, creado por Android Inc. y cuya finalidad es integrar el SDK de Android con Eclipse y proporcionar todas las herramientas necesarias para el desarrollo y la depuración de aplicaciones Android.

2.5.3. Weka para Android

Weka, según reza en su página web, es "una colección de algoritmos de aprendizaje automático para tareas de minería de datos. Contiene herramientas para el pre-procesamiento de los datos, la clasificación, la regresión, el *clustering*, las reglas de asociación, y la visualización" (17).

Entre sus algoritmos y aplicaciones dispone de una librería programada en Java para hacer uso de estas herramientas en programas que realice el usuario. Por desgracia, esta librería oficial no está adaptada para ser utilizada bajo el sistema operativo Android debido a que cuenta con una serie de código referente a la interfaz del programa que es incompatible.

Existe una librería de Weka, que es la que se utilizará en esta aplicación, que ha sido adaptada para su uso en Android eliminando estas incompatibilidades (18). Está basada en la versión 3 de la librería oficial de Weka y no se garantiza su completa funcionalidad, aunque las pruebas indican que para este proyecto sí es estable. Se encuentra disponible y de forma gratuita en el *hosting* de repositorios GitHub (19).

3. OBJETIVOS

Como se indicó en la introducción, este Trabajo Final de Grado pretende generar una aplicación para el sistema operativo Android que, mediante el uso de una red de neuronas, reconozca el patrón de actividades del usuario. Entre los múltiples patrones de actividad que puede realizar un usuario, se optó por acotar el resultado a 6 opciones: caminar, correr, estar sentado, estar de pie, subir escaleras y bajar escaleras.

Uno de los primeros problemas que se encontró a la hora de elegir los datos que se utilizarían para su clasificación, fue la necesidad que de dichos datos tuvieran una ventana de tiempo razonable. No se puede clasificar la actividad del usuario con tan solo ver un "fotograma" de su posición. Se requería una serie de capturas que indicaran el proceso de movimiento. Al final se decidió obtener 20 muestras de la posición del teléfono en 20 segundos.

En lo referente a los datos de entrenamiento, se optó en un principio por obtener los datos del GPS y el acelerómetro del teléfono, ya que se consideraron que serían la mejor influencia para que la clasificación sea la correcta. Después de realizar una serie de pruebas con estos datos y algunas variantes, se descubrió que los datos del GPS eran prácticamente irrelevantes para la clasificación, y se eliminaron del sistema.

La principal limitación que trae consigo un terminal *smartphone*, es su hardware. Debido a que son terminales de tamaño limitado, su hardware, como por ejemplo el procesador y la memoria RAM, también se ve reducido. Por ello, la elección de una red de neuronas ligera en cuanto al procesamiento es importante.

Como ya se ha especificado, es importante que la red neuronal sea rápida, pero más importante es la fiabilidad de la misma. Si una red neuronal no clasifica correctamente, es irrelevante que clasifique en un tiempo muy bajo. Al igual que con la velocidad, se realizaron varias pruebas con los datos de entrenamiento disponibles para buscar la mayor fiabilidad.

Comparando ambas pruebas en lo referente a velocidad y fiabilidad de la red neuronal, se decidió que la mejor opción a implementar sería un perceptrón multicapa, debido a que mostraba un porcentaje de acierto bastante bueno y una velocidad de carga de la red y clasificación de los datos rápida.

Ya especificado en apartados anteriores, el sistema operativo que soportará esta aplicación será Android, en su versión 2.3. Por tanto, uno de los principales objetivos es aprender a programar aplicaciones basadas en este entorno. Si bien es cierto que el lenguaje de programación es Java y ya se conoce en profundidad, muchas de las opciones que trae Android son desconocidas.

También se debe conocer y dominar el uso de los sensores del teléfono, principalmente el acelerómetro que será utilizado para captar los datos del movimiento del usuario. Es importante saber obtener los datos, pero también que el tiempo entre cada toma sea el apropiado y el acordado.

Una vez completados todos los objetivos anteriormente descritos, se procede al diseño y la programación de la aplicación correspondiente. En el diseño se debe aclarar cada aspecto de la

aplicación, como la interfaz de usuario, la forma de obtener los datos de los sensores, de almacenar dichos datos y de clasificar la información.

A nivel de programación, hay que cumplir los requisitos impuestos en el diseño. Es importante que se cumplan todos, pero también es importante que el código esté limpio y depurado de programación innecesaria y que cuente con los comentarios correspondientes para su completa comprensión. También se ha optado por mostrar un *log* que informe de la traza de ejecución de la aplicación, lo que ayudaría mucho a futuras actualizaciones y a saber dónde se producen los errores que vayan surgiendo.

Cuando se termina de programar cada uno de los apartados que componen la aplicación (acelerómetro, red de neuronas, almacenamiento…) se debe probar cada una de estas partes por separado, certificando su correcto funcionamiento. Cuando todo esté aceptado, se procede a la integración de las partes para las pruebas finales de todo el sistema.

Cada prueba debe ser documentada, estudiada y repetida hasta que el resultado sea el correcto. En caso de que no sea así, se deben analizar los resultados de las pruebas para corregir los errores existentes y que se cumplan los objetivos descritos.

Todos los objetivos anteriormente descritos en este apartado deben ser catalogados y documentados convenientemente. Una buena documentación permite a futuros usuarios que puedan modificar o ampliar las funcionalidades de la aplicación de una forma más sencilla.

4. DISEÑO DE LA APLICACIÓN

En este apartado se encuentra relatado todo lo referido al desarrollo real de la aplicación, desde los requerimientos impuestos por el usuario hasta la implementación final de la aplicación.

Al ser este un modelo en cascada, cada apartado detallado en esta sección es origen del siguiente y continuación del anterior, siguiendo una lógica descendente. Esto se traduce en que, a pesar de ser el modelo retroalimentado, en esta memoria no se apreciará tal característica, a pesar de que sí se ha utilizado durante el desarrollo real de la aplicación.

4.1. Requisitos de la aplicación

Los requisitos constituyen el punto de inicio de una aplicación, ya que son capaces de describir todo lo que el sistema será capaz de hacer (o no hacer), una vez terminado.

Cada uno de los requisitos (ya sean de usuario o de software) están recogidos en tablas con los campos necesarios para que el requisito contenga la información necesaria para su correcta comprensión, sin dar lugar a otros posibles alcances o soluciones. A continuación se detalla cada uno de los campos:

- **Identificador**: Clave unívoca que representa un requisito. Para cada uno de los posibles requisitos se utilizará la siguiente nomenclatura:
  - RU-Cxx: Requisito de Usuario del tipo Capacidad.
  - RU-Rxx: Requisito de Usuario del tipo Restricción.
  - RS-Fxx: Requisito de Software del tipo Funcional.
  - RS-Ixx: Requisito de Software del tipo Interacción.
  - RS-Rxx: Requisito de Software del tipo Rendimiento.
- **Título**: Nombre del requisito. Será complementado con la descripción para su comprensión.
- **Descripción**: Explicación del requisito. Deberá ser clara y concisa, evitando en todo momento cualquier duda o ambigüedad posible.
- **Necesidad**: Nivel de necesidad de que el requisito sea satisfecho.
- **Prioridad**: Nivel de prioridad del requisito a la hora de ser realizado.
- **Estabilidad**: Grado de variación que puede sufrir el requisito a lo largo de todo el trabajo.

4.1.1. Requisitos de usuario

Los requisitos de usuario definen aquellas características que el cliente demanda a la aplicación. Los requisitos que expresan capacidades del sistema se detallan a continuación:

Identificador	**RU-C01**		
Título	Funcionalidad principal		
Descripción	La aplicación deberá reconocer patrones de actividad que está realizando el usuario mientras éste lleva el teléfono en el bolsillo y la aplicación está en funcionamiento.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 2. Requisito RU-C01 - Funcionalidad principal.

Identificador	RU-C02		
Título	Estado actual		
Descripción	La aplicación mostrará en todo momento el estado del sistema, ya sea en ejecución o detenida.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 3. Requisito RU-C02 - Estado actual.

Identificador	RU-C03		
Título	Controles de reconocimiento		
Descripción	La aplicación dispondrá de una serie de botones que permitan detener el reconocimiento actual de patrones de actividad y poder reiniciarlo posteriormente.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 4. Requisito RU-C03 - Controles de reconocimiento.

Identificador	RU-C04		
Título	Finalización total		
Descripción	La aplicación contará con un botón para que ésta finalice completamente sin dejar procesos en ejecución.		
Necesidad	☐ Esencial	☑ Deseable	☐ Opcional
Prioridad	☐ Alta	☑ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 5. Requisito RU-C04 - Finalización total.

Los requisitos que expresan restricciones del sistema se detallan a continuación:

Identificador	RU-R01		
Título	Actividades reconocidas		
Descripción	La aplicación reconocerá exclusivamente los siguientes patrones de actividad: estar sentado, estar de pie, caminar, correr, subir escaleras y bajar escaleras.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 6. Requisito RU-R01 - Actividades reconocidas.

Identificador	RU-R02		
Título	Tipos de estado		
Descripción	La aplicación mostrará el estado de "detenida" mediante la palabra correspondiente, y el estado "en ejecución" mediante el resultado del último reconocimiento.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 7. Requisito RU-R02 - Tipos de estado.

Identificador	RU-R03		
Título	Tiempo de carga		
Descripción	La aplicación se cargará en un tiempo inferior a 15 segundos.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 8. Requisito RU-R03 - Tiempo de carga.

Identificador	RU-R04		
Título	Tiempo de ejecución		
Descripción	La aplicación ejecutará las acciones necesarias en un tiempo lo suficientemente pequeño como para que el reconocimiento se realice en tiempo real y se muestre antes del siguiente reconocimiento.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 9. Requisito RU-R04 - Tiempo de ejecución.

Identificador	RU-R05		
Título	Versión del sistema Android		
Descripción	La aplicación será desarrollada para la versión 2.3 de Android, pudiendo ser utilizada en versiones posteriores.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 10. Requisito RU-R05 - Versión del sistema Android.

4.1.2. Requisitos de software

Los requisitos de software definen cual es el comportamiento del sistema en base a lo extraído de los requisitos de usuario redactados anteriormente, y son los siguientes:

Identificador	RS-F01		
Título	Red de neuronas artificiales		
Descripción	La aplicación contará con una red de neuronas artificiales para el reconocimiento de patrones de actividad del usuario.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 11. Requisito RS-F01 - Red de neuronas artificiales.

Identificador	RS-F02		
Título	Utilización del acelerómetro		
Descripción	La aplicación hará uso del acelerómetro del teléfono para reconocer el patrón de actividades del usuario.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 12. Requisito RS-F02 - Utilización del acelerómetro.

Identificador	RS-F03		
Título	Almacenamiento de los datos del acelerómetro		
Descripción	La aplicación contará con un sistema de almacenamiento dinámico que contenga los datos que se han ido capturando del acelerómetro.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 13. Requisito RS-F03 - Almacenamiento de los datos del acelerómetro.

Identificador	RS-F04		
Título	Estado actual de la aplicación		
Descripción	La aplicación mostrará en todo momento el estado actual de la misma, ya sea porque está en proceso de reconocimiento o porque está detenida.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 14. Requisito RS-F04 - Estado actual de la aplicación.

Identificador	RS-I01		
Título	Botón para detener el reconocimiento		
Descripción	La aplicación contará con un botón que permita detener el proceso de reconocimiento de patrones de actividad del usuario.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 15. Requisito RS-I01 - Botón para detener el reconocimiento.

Identificador	RS-I02		
Título	Botón para iniciar el reconocimiento		
Descripción	La aplicación contará con un botón que permita iniciar el proceso de reconocimiento de patrones de actividad del usuario.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 16. Requisito RS-I02 - Botón para iniciar el reconocimiento.

Identificador	RS-I03		
Título	Botón para salir de la aplicación		
Descripción	La aplicación contará con un botón que permita finalizar completamente la aplicación sin dejar procesos en ejecución.		
Necesidad	☐ Esencial	☑ Deseable	☐ Opcional
Prioridad	☐ Alta	☑ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 17. Requisito RS-I03 - Botón para salir de la aplicación.

Identificador	RS-I04		
Título	Diálogo de confirmación para salir		
Descripción	La aplicación contará con un diálogo que solicite la confirmación del usuario para salir de la aplicación.		
Necesidad	☐ Esencial	☐ Deseable	☑ Opcional
Prioridad	☐ Alta	☐ Media	☑ Baja
Estabilidad	☐ Alta	☑ Media	☐ Baja

Tabla 18. Requisito RS-I04 - Diálogo de confirmación para salir.

Identificador	RS-R01		
Título	Actividades reconocidas		
Descripción	La aplicación reconocerá exclusivamente los siguientes patrones de actividad: estar sentado, estar de pie, caminar, correr, subir escaleras y bajar escaleras.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 19. Requisito RS-R01 - Actividades reconocidas.

Identificador	RS-R02		
Título	Tipos de estado		
Descripción	La aplicación mostrará el estado “detenida” mediante la palabra “Detenida”, y el estado “en ejecución” mediante las palabras “Estar sentado”, “Estar de pie”, “Caminar”, “Correr”, “Subir escaleras” y “Bajar escaleras”, correspondientes al resultado del último reconocimiento.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 20. Requisito RS-R02 - Tipos de estado.

Identificador	RS-R03		
Título	Estado inicial del reconocimiento		
Descripción	La aplicación mostrará la palabra "Inicializando" cuando se comience el reconocimiento de patrones de actividad del usuario y no se haya obtenido aún un resultado de reconocimiento.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 21. Requisito RS-R03 - Estado inicial del reconocimiento.

Identificador	RS-R04		
Título	Tiempo de carga de la aplicación		
Descripción	La aplicación deberá cargar en un tiempo inferior a 15 segundos, momento en el cual estará operativa para el usuario.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 22. Requisito RS-R04 - Tiempo de carga de la aplicación.

Identificador	RS-R05		
Título	Red de neuronas precargada		
Descripción	La aplicación contará con una red de neuronas precargada y entrenada para mejorar el tiempo de carga del sistema.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☐ Alta	☑ Media	☐ Baja

Tabla 23. Requisito RS-R05 - Red de neuronas precargada.

Identificador	RS-R06		
Título	Tiempo de ejecución		
Descripción	La aplicación deberá mostrar el resultado del último reconocimiento de patrones de actividad del usuario antes de que el siguiente reconocimiento se lleve a cabo.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 24. Requisito RS-R06 - Tiempo de ejecución.

Identificador	RS-R07		
Título	Versión mínima de Android		
Descripción	La aplicación se desarrollará para ser ejecutada en la versión 2.3 de Android, pudiendo ser utilizada en posteriores versiones. No se garantiza su funcionamiento en versiones anteriores.		
Necesidad	☑ Esencial	☐ Deseable	☐ Opcional
Prioridad	☑ Alta	☐ Media	☐ Baja
Estabilidad	☑ Alta	☐ Media	☐ Baja

Tabla 25. Requisito Rs-R07 - Versión mínima de Android.

4.1.3. Matriz de trazabilidad

Una vez que se han obtenido todos los requisitos, se debe comprobar que existe una continuidad entre lo que ha demandado el usuario, y las características que realmente se han concluido que tendrá la herramienta.

Para ello es necesario elaborar una matriz de trazabilidad, en la que se unen requisitos de usuario y de software, de forma que un requisito de usuario debe estar relacionado con, al menos, uno o más requisitos de software.

La matriz de trazabilidad de la Tabla 26 ha comprobado lo dicho en el párrafo anterior, obteniéndose un resultado satisfactorio. Esto quiere decir que no se deben retocar los requisitos por problemas de cohesión en el desarrollo, lo que no implica que no se pueda modificar o añadir algún requisito en el futuro por otras causas.

	RU-C01	RU-C02	RU-C03	RU-C04	RU-R01	RU-R02	RU-R03	RU-R04	RU-R05
RS-F01	×								
RS-F02	×								
RS-F03	×								
RS-F04		×							
RS-I01			×						
RS-I02			×						
RS-I03				×					
RS-I04				×					
RS-R01					×				
RS-R02						×			
RS-R03						×			
RS-R04							×		
RS-R05							×		
RS-R06								×	
RS-R07									×

Tabla 26. Matriz de trazabilidad entre los requisitos de usuario y de software.

4.2. Casos de uso

Un caso de uso es una secuencia de interacciones que se desarrollan entre el sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. En este apartado se van a describir las secuencias de uso que se ejecutarán en el sistema. Se detallarán de forma textual cada uno de los casos de uso.

Por regla general se incluye el tipo de actor que realizará el caso de uso, pero en este caso se ha eliminado dicho dato debido a que no existen diferentes tipos de actores y siempre será el usuario final. En la descripción textual de cada uno de los casos de uso, se especificarán los campos que se muestran a continuación:

- **Identificador**: Determinará de forma unívoca cada uno de los casos de uso que se detallarán a continuación. El formato a seguir será: CU-XX siendo XX un número secuencial que comenzará por el 01 e irá incrementando una unidad por cada caso de uso nuevo.

- **Título**: Nombre descriptivo acerca del caso de uso.
- **Descripción**: Breve descripción del caso de uso a comentar.
- **Precondiciones**: Condiciones previas que se deberán cumplir para poder ejecutar el caso de uso.
- **Postcondiciones**: Condiciones que se producirán tras la ejecución del caso de uso.
- **Escenario principal**: Trata de la secuencia común de interacciones ordenadas, especificando la interacción del usuario con el sistema.
- **Escenario alternativo**: Describirá la ejecución del caso de uso con condiciones de error o caminos de decisión distintos al principal.

Los casos de uso textuales son los siguientes:

Identificador	**CU-01**
Título	Iniciar aplicación
Descripción	El usuario iniciará la aplicación para su posterior uso.
Precondiciones	La aplicación estaba cerrada.
Postcondiciones	La aplicación estará iniciada y detenida.
Escenario principal	1. El usuario hace clic en el botón de acceso a la aplicación. 2. La aplicación se enciende. 3. La aplicación muestra una barra de carga. 4. La aplicación carga la red neuronal, el acelerómetro y el almacenamiento de datos. 5. La aplicación muestra la ventana de inicio.
Escenario alternativo	4a. Se produce un problema al cargar la red neuronal, el acelerómetro, o el almacenamiento de datos. 5. La aplicación muestra una ventana de error de inicio.

Tabla 27. Caso de uso CU-01 – Iniciar aplicación.

Identificador	CU-02
Título	Iniciar reconocimiento de patrones de actividad
Descripción	El usuario iniciará el reconocimiento de patrones de actividad del usuario para que la aplicación reconozca el patrón que realizará desde ese momento.
Precondiciones	La aplicación estaba iniciada y detenida.
Postcondiciones	La aplicación estará iniciada y en modo de reconocimiento.
Escenario principal	1. El usuario hace clic en el botón de "Empezar". 2. La aplicación cambia el estado de la interfaz a "Inicializando". 3. El usuario introduce su teléfono en el bolsillo. 4. La aplicación comienza a capturar datos del acelerómetro de forma periódica y a almacenarlos en el almacenamiento dinámico. 5. La aplicación comienza a utilizar los datos almacenados y a procesarlos mediante la red neuronal de forma periódica. 6. La aplicación, ante cada reconocimiento, muestra en la interfaz el resultado del último reconocimiento del patrón de actividad del usuario.
Escenario alternativo	4a. Se produce un problema al capturar los datos del acelerómetro. 5. La aplicación muestra una ventana de error.
Escenario alternativo	4a. Se produce un error al almacenar los datos en el almacenamiento dinámico. 5. La aplicación muestra una ventana de error.
Escenario alternativo	5a. Se produce un error al ejecutar la red neuronal con los datos obtenidos del almacenamiento dinámico. 6. Se muestra un mensaje de error en el procesamiento de la red neuronal.

Tabla 28. Caso de uso CU-02 - Iniciar reconocimiento de patrones de actividad.

Identificador	CU-03
Título	Detener reconocimiento de patrones de actividad
Descripción	El usuario detendrá el reconocimiento de patrones de actividad del usuario.
Precondiciones	La aplicación estaba iniciada y en modo de reconocimiento.
Postcondiciones	La aplicación estará iniciada y detenida.
Escenario principal	1. El usuario hace clic en el botón de "Detener". 2. La aplicación cambia el estado de la interfaz a "Detenida". 3. La aplicación detiene la captura de datos del acelerómetro y su almacenamiento en el almacenamiento dinámico. 4. La aplicación detiene el uso de los datos almacenados y su procesamiento mediante la red neuronal de forma periódica.

Tabla 29. Caso de uso CU-03 - Detener reconocimiento de patrones de actividad.

Identificador	CU-04
Título	Finalizar aplicación
Descripción	El usuario finaliza la aplicación.
Precondiciones	La aplicación estaba iniciada.
Postcondiciones	La aplicación estará cerrada.
Escenario principal	1. El usuario hace clic en el botón Atrás del teléfono. 2. La aplicación muestra un diálogo de confirmación. 3. El usuario hace clic en el botón "Si". 4. La aplicación finaliza completamente.
Escenario alternativo	3a. El usuario hace clic en el botón "No". 4. Se cierra el cuadro de diálogo y se mantiene el estado actual de la aplicación.

Tabla 30. Caso de uso CU-04 - Finalizar aplicación.

4.3. Arquitectura de la aplicación

Después de haber definido qué es lo que el usuario desea obtener de esta aplicación y haber visto ejemplos prácticos de uso de la futura aplicación, es el momento de que se comience la creación del sistema, realizando un diseño de la aplicación a alto nivel.

Este sistema, en base a lo observado en las peticiones del cliente y a las características del sistema operativo en el que se va a desarrollar, será un sistema que contendrá interfaces con las que el usuario interactuará y una parte dedicada solo a la lógica del sistema y el tratamiento de la aplicación con el sistema operativo, por lo que para la interacción entre las interfaces y la lógica, este sistema necesitará otra pequeña parte dedicada a la comunicación entre las interfaces y la lógica, haciendo de puente entre ellas dos.

Visto todo esto, y una vez estudiados los patrones más conocidos y utilizados en la ingeniería del software, se comprueba que el patrón más acorde a este sistema sería una arquitectura Modelo-Vista-Controlador (MVC). Este patrón está formado por tres partes diferenciadas:

- **Modelo**: Es el encargado de tratar toda la lógica del negocio, representando específicamente toda la información con la que el sistema va a trabajar y notificando a la vista los cambios que se producen en el sistema. Es el "motor" de la aplicación.
- **Vista**: Es la parte del sistema con la que interacciona el usuario, recibiendo y mostrando datos en pantalla. Normalmente se cuenta con más de una interfaz, por lo que también se encarga de navegar entre las distintas interfaces. Es la "cara" de la aplicación.
- **Controlador**: Procesa las peticiones que realiza el usuario y las envía a la lógica del sistema para que las trate y devuelva un resultado la información adecuada para mostrársela al usuario. Es el "transportador" de la aplicación.

En la Ilustración 8 se puede observar la arquitectura MVC y la conexión que existen entre cada una de sus partes:

Ilustración 8. Diagrama del sistema Modelo-Vista-Controlador.

Para que el entendimiento del funcionamiento de la arquitectura del sistema sea más sencillo, se muestra el siguiente ejemplo genérico:

1. El usuario desea ejecutar una actividad cualquiera de la aplicación, por lo que realiza la acción determinada en la pantalla del *smartphone*.
2. La vista que se esté ejecutando en ese momento identifica el evento que ha ocurrido y se lo notifica al controlador, mandándole además los datos que necesite.
3. El controlador se comunica con el modelo, pasándole los datos que ha recibido de la vista.
4. El modelo, con la información recibida, ejecuta todos los comandos y acciones que sean necesarias, para, a continuación, enviar una respuesta al controlador.
5. El controlador recibe la respuesta enviada por el modelo y se la da a la vista en formato legible para el usuario.
6. La vista recibe la respuesta dada y se la muestra al usuario, cambiando incluso de interfaz si fuera necesario.

4.4. Matriz de trazabilidad de la arquitectura

Esta arquitectura, como se comentó en su momento, debe cumplir con la premisa de que todo requisito de software debe encontrarse en, al menos, un componente de la arquitectura, por lo que es necesario desarrollar un mecanismo de comprobación. La matriz de trazabilidad de la Tabla 31 confirma esto:

	Modelo	Vista	Controlador
RS-F01	×		
RS-F02	×		
RS-F03		×	
RS-F04		×	
RS-I01			×
RS-I02			×
RS-I03			×
RS-I04		×	×
RS-R01	×		
RS-R02	×		
RS-R03	×		
RS-R04	×		
RS-R05	×		
RS-R06	×		
RS-R07	×		

Tabla 31. Matriz de trazabilidad entre los requisitos de software y el modelo MVC.

5. IMPLEMENTACIÓN DEL SISTEMA

En este apartado se tratará todo el proceso de codificación real del sistema, describiendo el lenguaje de programación usado y el entorno de programación en el que se ha desarrollado la aplicación, mostrando el diagrama de clases y explicando las características fundamentales de cada una y, por último, explicando de forma detallada cada una de las decisiones importantes tomadas a la hora de la implementación en código.

5.1. Lenguaje de programación

Android, que es el sistema operativo donde se va a desarrollar esta aplicación, utiliza como lenguaje de programación el conocido lenguaje Java. Este es un lenguaje de alto nivel, y a pesar de que la mayoría de aplicaciones para Android son realizadas en Java (por la facilidad de desarrollar en alto nivel frente al bajo nivel), conviene saber que los archivos y librerías de Android se encuentran codificadas en lenguaje C/C++, las cuales son compiladas en código nativo ARM y ejecutadas por el sistema operativo.

Para el desarrollo de aplicaciones para Android, Google (propietario del sistema operativo) ofrece a todos los programadores un kit de desarrollo, el Android SDK (ya que Android es un sistema operativo de código abierto). Este kit, contiene (tal y como se ha descrito en el punto 2.5.1): las herramientas para el desarrollo del código; complementos como la librería para trabajar con Google Maps; drivers para Windows que permiten la ejecución del código en un dispositivo mediante USB; ejemplos de código para programadores principiantes; documentación con la última API de Android.

Google, como se acaba de comentar, también otorga a los desarrolladores una API muy moderna y completa (incluso más que la propia API de Java), donde se puede encontrar desde todos los atributos y métodos disponibles de Android, hasta gran cantidad de ejemplos con explicaciones de cómo crear clases, interfaces, servicios…

5.2. Entorno de desarrollo

El lenguaje de programación es Java. Bajo el punto de vista del creador (y de gran parte de la comunidad desarrolladora), para el desarrollo de código en lenguaje Java existen dos principales entornos de desarrollo: NetBeans y Eclipse. Cualquiera de estos dos hubiera sido útil, pero fue elegido Eclipse (versión Classic 3.7.2) por la otra gran razón descrita a continuación.

El kit de desarrollo SDK contiene un *plug-in* para Eclipse llamado ADT (Android Development Tools) (20). Esta fue la principal razón para elegir Eclipse en vez de NetBeans. Este *plug-in* crea una configuración de proyecto de tal forma que sólo hay que preocuparse por la codificación del sistema, ya que se encarga de preparar el entorno con una configuración de proyecto que permite compilar y ejecutar de forma similar a la que se haría si fuera un proyecto Java únicamente.

5.3. Diagrama de clases

En un apartado anterior se mostró la arquitectura del sistema, lo que daba lugar a un primer esbozo a muy alto nivel de cuál sería la estructura del sistema: tres grandes bloques, cada uno

con una función bien definida. Este modelo lógico a alto nivel, se ha ido detallando cada vez más, hasta llegar a un diseño a nivel de clases, utilizando para ello el lenguaje UML (21).

El diagrama de clases UML muestra las clases del sistema con sus atributos y métodos, así como sus relaciones estructurales y de herencia existentes entre ellas. En la Ilustración 9 se muestra el diseño de clases completo, con sus atributos y métodos así como las relaciones entre los mismos.

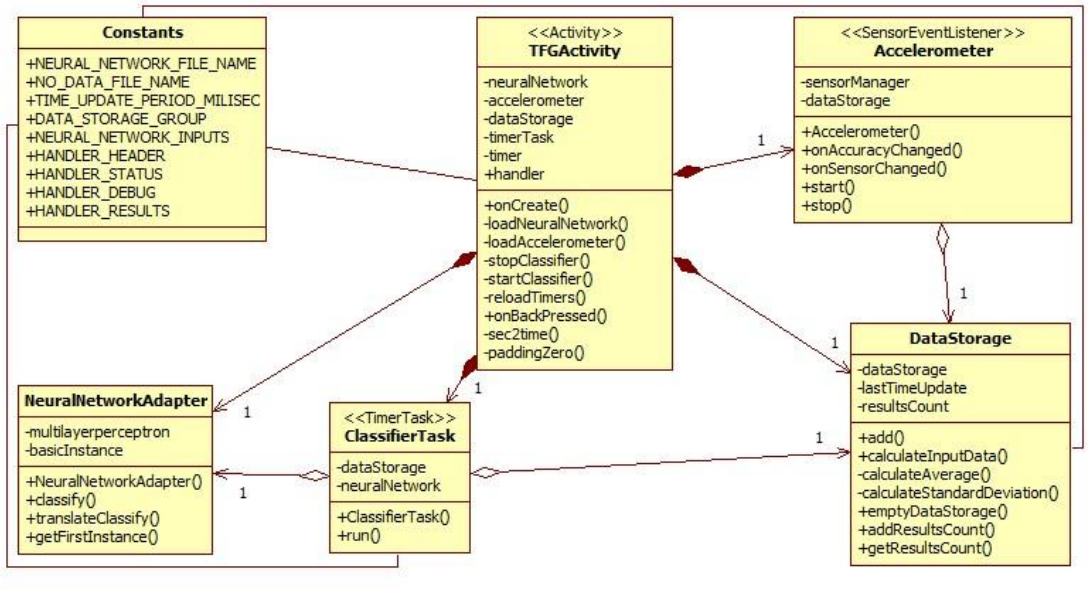


Ilustración 9. Diagrama de clases.

En el diagrama UML, se puede observar todas las clases Java que pertenecen al desarrollo de la aplicación, pero sin incluir aquellas clases que pertenecen al desarrollo propio de Android, como serían los documentos XML para el desarrollo de las interfaces, el *AndroidDocument* o la clase R.java.

Continuando en el punto del desarrollo donde se encontraba el proyecto, se concluyó que el diseño arquitectónico seguiría el patrón MVC (Modelo-Vista-Controlador), con tres partes claramente diferenciadas: las interfaces, la lógica y la comunicación entre ambas partes:

- La interfaz, también conocida como Vista, es el fichero XML que contiene toda la estructura de la misma, el *layout*.
- La comunicación o Controlador corre a parte de la clase *TFGActivity* que, como ya se ha indicado, sirve de puente entre la interfaz y la lógica de la aplicación.
- La lógica de la aplicación o Modelo, se corresponde con el resto de clases detalladas en el diagrama anterior y contienen toda la lógica y el modelo necesario para ejecutar la aplicación.

Todo esto indica, de forma clara, que no existe ningún tipo de comunicación real entre el paquete “Modelo” y el paquete “Vista”, respetando una de las ideas principales de este patrón.

5.4. Decisiones de implementación

Durante el desarrollo de la aplicación normalmente surgen incógnitas que no habían aparecido antes o que simplemente sólo pertenecen a este punto del desarrollo, y que son solventadas tomando decisiones que únicamente entiende el equipo de desarrollo. Estas decisiones deben ser, por tanto, explicadas para que todo aquel que estudie el código o simplemente hago uso de la aplicación, entienda el por qué.

En este apartado se van a explicar las decisiones más importantes tomadas durante la implementación, y que se han dividido en base a las características principales que posee el sistema.

5.4.1. Interfaz

A la hora de realizar una aplicación, ya sea para dispositivos móviles o para cualquier otro tipo de sistema, se debe tener muy en cuenta la interfaz del sistema, ventanas y demás opciones.

Para este proyecto se optó por una simple ventana con dos botones que cubriría toda la aplicación, ya que lo único que el usuario requería era una aplicación que le mostrara la actividad que está realizando. Esta interfaz cuenta con dos botones en la parte inferior para "Empezar" y "Detener" el proceso de reconocimiento de patrones de actividad, un cuadro de texto para mostrar la actividad actual del usuario y una pequeña lista del tiempo que ha realizado cada actividad.

En el sistema Android, las interfaces de la aplicación son conocidas como *Activity*. Cada una de las *Activity* corresponde con una ventana de la aplicación, que a su vez corresponde con dos ficheros dentro del proyecto de código. El primer fichero, con formato XML y denominado *layout* contiene la distribución de la ventana y de sus partes, el texto inicial de la ventana y todos los elementos de los que dispondrá y su personalización (tipo de letra, tamaños, márgenes…). El otro fichero, en formato Java, contiene toda la lógica de la ventana, captura de botones pulsados, cambios de ventanas (que aquí no se utilizan) y todo lo necesario para que la ventana pueda funcionar. Claramente se separa la estructura con la funcionalidad.

Como ya se ha indicado antes, esta aplicación tan solo dispone de una ventana. A continuación se muestra un ejemplo de dicha ventana:

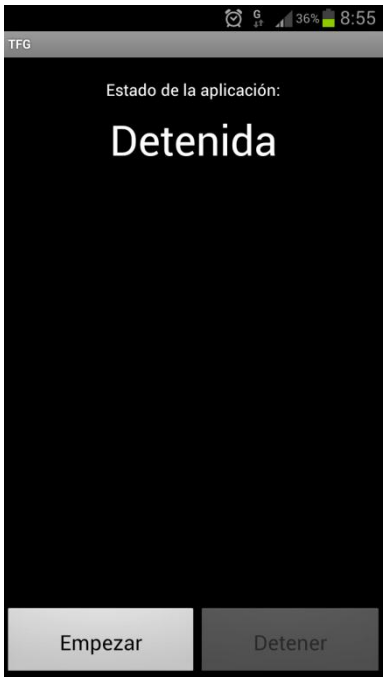


Ilustración 10. Ejemplo de interfaz en el estado inicial.

Como puede observarse en la Ilustración 10, abajo hay dos botones para activar y desactivar el reconocimiento de patrones de actividad del usuario. El de "Empezar" está habilitado ya que el reconocimiento está desactivado, mientras que el de "Detener" permanecerá deshabilitado para cuando lo necesite el usuario. En caso de comenzar con el proceso de reconocimiento, la vista pasará a tener la estructura de la Ilustración 11.

Ilustración 11. Ejemplo de interfaz en proceso de reconocimiento.

Como es obvio, mientras el proceso esté activado, se invertirán los botones, pasando a desactivar el de "Empezar" y habilitar el de "Detener". En el momento en el cual el usuario pulse sobre el botón detener para finalizar el proceso, este puede ver sus resultados como se observa en la Ilustración 12.

Ilustración 12. Ejemplo de interfaz una vez detenido el reconocimiento.

En la parte superior de la ventana se muestra el último reconocimiento obtenido de la red de neuronas. Justo debajo aparecen tres números, correspondientes a la última lectura del acelerómetro del terminal. Y, para terminar, en la parte central de la ventana se muestra una lista de tiempos para que el usuario conozca la duración de cada una de sus actividades.

Cuando se pulsa el botón de atrás, botón destinado al cierre total de la aplicación, se le solicita al usuario que confirme la acción por si ha pulsado el botón por error (ver Ilustración 13). Una vez respondida la pregunta de confirmación, se procede a salir de la aplicación o a continuar con su tarea actual (reconocimiento o detenido).

Ilustración 13. Ejemplo de interfaz con cuadro de confirmación para salir.

Como pequeño detalle, se ha optado por hacer que la aplicación no sea a pantalla completa debido a que no requiere de mucho espacio y permite al usuario tener siempre disponible la barra superior.

La clase Java que contiene toda la lógica de la ventana se llama *TFGActivity*. En su interior se encuentran todos los objetos de la red neuronal, el almacenamiento de datos, la tarea en paralelo… Todo ello se explicará más adelante.

5.4.2. Ejecución en paralelo

Android, como sistema operativo móvil, ofrece la capacidad de continuar ejecutando aplicaciones en un segundo plano y ejecutar múltiples hilos de forma paralela. Para ello, se optó por crear un hilo de ejecución paralelo, característica proporcionada por Java y soportada sin problemas en Android. Esta clase que ejecuta parte del código fuente se conoce como *Thread* y requiere de un *TimerTask* para su ejecución, que es una clase que ejecuta *Threads* de forma cíclica cada X tiempo.

La principal cualidad de esta ejecución en paralelo es que permite indicar el plazo de tiempo que debe pasar entre el inicio de su ejecución y la siguiente, lo cual coincide perfectamente con nuestras necesidades. Además, esta ejecución en paralelo, permite que las acciones que realice el usuario (pulsar los botones) se realicen en el momento, mientras que si no existiera la ejecución en paralelo, habría que esperar a que finalizase todo el código para poder capturar ese evento.

Entrando un poco más en detalle, la clase Java que permite una ejecución en paralelo es la clase *TimerTask*, que ha derivado mediante la herencia en la clase *ClassifierTask*. Esta última es la que se ha desarrollado, mientras que la primera es parte de la API de Java. Esta clase contiene el objeto *DataStorage* para el almacenamiento de datos y el objeto *NeuralNetwork* que contiene la red neuronal. Ambos se explicarán más adelante.

5.4.3. Almacenamiento de datos

Esta aplicación requiere una pequeña parte de la memoria para almacenar los datos recibidos del acelerómetro que, posteriormente, se transmitirán a la red neuronal. La clase *DataStorage* contiene toda la lógica necesaria para el almacenamiento, procesamiento y distribución de los datos. Contiene también una lista a modo de contador para acumular la cantidad de veces que se realiza cada acción.

Merece una pequeña mención la función *CalculateInputData* dentro de esta clase. Dicha función es llamada por el proceso en paralelo y calcula, gracias a los datos almacenados en la lista, todos los resultados que se deben proveer a la red neuronal para su procesamiento.

5.4.4. Acelerómetro

El acelerómetro es una de las partes cruciales de esta aplicación, ya que provee a la red de neuronas los datos necesarios para que esta los procese y devuelva un resultado. La clase *Accelerometer* contiene todo lo necesario para este proceso. Dispone de un *Manager* que captura el sensor, un *Listener* que queda a la espera de que dicho sensor devuelva los datos y un *DataStorage* para almacenar los datos obtenidos.

Como la captura de datos no sigue una secuencia temporal definida, se ha optado por configurar el sensor para que devuelva los datos del acelerómetro lo más rápido posible. Luego, dentro del propio código, se comprueba si ha pasado un segundo para almacenar esos datos o descartarlos.

5.4.5. Red de neuronas artificiales

Gracias a las librerías de Weka, en Java se dispone de múltiples redes neuronales, clasificadores y demás para su uso en programas o aplicaciones, como es el caso. La clase *NeuralNetworkAdapter* es un adaptador que enlaza la aplicación y los datos del acelerómetro con la red neuronal. Esta clase contiene un objeto *MultilayerPerceptron* que representa al perceptrón multicapa utilizado como red neuronal en este proyecto.

Entre sus funciones, una de las más importantes es el constructor, que es la función que se ejecuta cuando se crea el objeto del adaptador. Este constructor se encarga de leer de un fichero la red de neuronas previamente entrenada para ser cargada directamente en la aplicación, sin necesidad de realizar el proceso de entrenamiento que es muy largo y costoso.

Otra de las funciones más importantes es *Classify*. Esta función recibe los datos previamente almacenados en el *DataStorage*, los traslada a una instancia (que es el objeto que contiene los datos para su procesamiento en redes neuronales) y se los pasa a la red neuronal a la espera de un resultado. Cuando la red neuronal devuelve el resultado, esta función transporta el resultado a la clase desde la cual ha sido llamada.

5.4.6. Constantes

Como última clase del proyecto tenemos la clase *Constants*, que contiene todas las constantes utilizadas en el proyecto. Para ser preciso, son estas:

- *NEURAL_NETWORK_FILE_NAME* contiene el nombre del fichero que, a su vez, contiene a la red neuronal ya entrenada.
- *NO_DATA_FILE_NAME* contiene el nombre de un fichero que, a su vez, contiene una instancia vacía, sin datos.
- *TIME_UPDATE_PERIOD_MILISEC* contiene el tiempo, en milisegundos, que debe pasar entre cada una de las ejecuciones del proceso en paralelo que, a su vez, ejecuta a la red neuronal.
- *DATA_STORAGE_GROUP* contiene la cantidad de datos que serán almacenados en la memoria de la aplicación y deben ser utilizados para enviarlos a la red neuronal.
- *NEURAL_NETWORK_INPUTS* contiene el número de neuronas de entrada de la red neuronal.
- *HANDLER_HEADER*, *HANDLER_STATUS*, *HANDLER_DEBUG* y *HANDLER_RESULTS* son unos números que se utilizan para la actualización de la cabecera de la interfaz, el estado actual de la clasificación, los datos del acelerómetro calculados y los resultados de la clasificación, respectivamente.

6. PRUEBAS Y EVALUACIÓN

En este capítulo se van a presentar las pruebas realizadas al sistema, con las cuales aseguraremos que los requisitos definidos al principio del proyecto se han cumplido. En un primer punto se describirá el entorno de pruebas donde se han realizado, para continuar en los siguientes puntos con las pruebas llevadas a cabo. En el último punto de este apartado se hará un pequeño resumen de los resultados obtenidos, destacando los puntos más importantes de las pruebas.

6.1. Descripción del conjunto de datos

Para el entrenamiento de la red neuronal, se ha utilizado un fichero de instancias referentes a los datos capturados por el acelerómetro y transformados a los datos que recibe la red neuronal. Este fichero cuenta con 5714 tuplas que representan los diferentes casos que pueden darse en el reconocimiento de patrones de actividad del usuario: estar sentado, estar de pie, caminar, correr, subir escaleras y bajar escaleras. Los datos se obtuvieron previamente al desarrollo de la aplicación mediante otra aplicación diferente cuyo único propósito era la recolección de datos.

	Estar sentado	Estar de pie	Caminar	Correr	Subir escaleras	Bajar escaleras
Total de tuplas	677	308	1722	2636	188	183
Porcentaje	11,85%	5,39%	30,14%	46,13%	3,29%	3,20%

Tabla 32. Número y porcentaje de tuplas para cada actividad.

Como puede observarse en la Tabla 32, existe un desbalanceo en el total de tuplas para cada actividad. Mientras que correr y caminar son las actividades más reconocidas (realizadas durante mayor tiempo en el proceso de recolección de datos), subir y bajar escaleras quedan con un menor número de ejemplos. Tal desbalanceo surge debido a la naturaleza de las mismas actividades, ya que unas son mucho más comunes que otras.

En ciertos casos, este desbalanceo puede dar lugar a un entrenamiento incorrecto de la red neuronal, ya sea por falta de ejemplos, por el propio desbalanceo, o simplemente porque la red de neuronas se adapta a este grupo de ejemplos sin representar el total que puede darse. Para ello, se realizarán pruebas de validación del conjunto de datos, comprobando que dicho modelo se ajusta a la realidad y no ofrece problemas.

6.2. Descripción del entorno de pruebas

Para la realización de las pruebas se han utilizado varios *smartphone* con sistema operativo Android. Cada uno de ellos posee características diferentes para corroborar que los requisitos se cumplen en todo tipo de terminales. Los teléfonos utilizados han sido los siguientes:

- **Samsung Galaxy SIII - I9300**
  - *Sistema Operativo*: Android 4.0.4 (Ice Cream Sandwich)
  - *Procesador*: Exynos 4 Quad Core 1,4 GHz, GPU Mail 400MP
  - *Memoria RAM*: 1 GB
  - *Tamaño*: 136,6 x 70,6 x 8,6 mm
  - *Peso*: 133 gramos
  - *Pantalla*: Táctil capacitiva de 4,8 pulgadas (1280 x 720)

- **LG Optimus Black**
  - *Sistema Operativo*: Android 2.3 (Gingerbread)
  - *Procesador*: TI OMAP 3630 1GHz
  - *Memoria RAM*: 512 MB
  - *Tamaño*: 122 x 64 x 9,2 mm
  - *Peso*: 109 gramos
  - *Pantalla*: Táctil capacitiva de 4,0 pulgadas (800 x 480)

- **Orange Monte Carlo**
  - *Sistema Operativo*: Android 2.3 (Gingerbread)
  - *Procesador*: 800 MHz
  - *Memoria RAM*: 512 MB
  - *Tamaño*: 126 x 67 x 11,3 mm
  - *Peso*: 140 gramos
  - *Pantalla*: Táctil capacitiva de 4,3 pulgadas (800 x 480)

6.3. Validación

En este apartado se explican cada una de las pruebas realizadas al sistema, para comprobar el correcto funcionamiento de la aplicación. Se realizaron inicialmente unas pruebas unitarias de cada parte de la aplicación, para posteriormente realizar pruebas conjuntas de todo el sistema, de forma que se pudiera abarcar la mayor posibilidad de entornos distintos, ya que cada una está orientada a un objetivo de comprobación distinto.

6.3.1. Prueba unitaria de la red de neuronas

Para probar la red de neuronas que se implementó en la aplicación, se realizaron dos pruebas diferentes. La primera consistía en reconocer las instancias del fichero de forma independiente y calcular el porcentaje de éxito. La segunda se valía del método de validación cruzada para una mayor fiabilidad de los resultados en cuanto al porcentaje de éxito.

Las pruebas unitarias revelaron un porcentaje de acierto de, aproximadamente, el 97,5% del total de casos comprobados. Además, esta prueba se realizaba en un plazo de tiempo mínimo, lo que casi garantizaba que la aplicación funcionaría en tiempo real.

La validación cruzada mostró idénticos resultados a los proporcionados por el programa Weka y que se pueden ver en la sección 6.3.4, por lo que se determinaba que la red neuronal había sido programada correctamente de acuerdo a las especificaciones previas.

6.3.2. Prueba unitaria del acelerómetro

Para comprobar que el acelerómetro funcionaba correctamente, se conectaron los dispositivos utilizados en las pruebas mediante un cable USB a un ordenador. La aplicación destinada a esta prueba mostraba a través del *log* de Android los datos capturados por el acelerómetro junto a la fecha y hora de captura. La conexión USB permitía que dichos datos pudieran verse en el ordenador mediante el programa Eclipse y el *plug-in* ADT de Android.

Una de las pruebas capturaba los datos lo más rápido posible, mientras que la otra capturaba los datos del acelerómetro una vez por segundo. Comprobando el resultado de los *logs* se pudo determinar que el acelerómetro funcionaba correctamente.

6.3.3. Pruebas unitaria del almacenamiento de datos

En la validación del almacenamiento dinámico de los datos de la aplicación, se realizaron pruebas con el propio acelerómetro, ya que éste había sido validado correctamente. Las pruebas consistían en introducir los datos en el almacenamiento según se iban recibiendo en el acelerómetro, y mostrar dichos datos mediante el *log*.

Dado que el almacenamiento de datos en listas es una cosa sencilla, no hubo ningún problema y se validó este sistema en la primera prueba con perfectos resultados.

6.3.4. Pruebas de validación del modelo de datos

Una vez que se ha logrado que las partes de la aplicación funcionen por separado, llega uno de los puntos más críticos de este proyecto: comprobar que el modelo de datos del cual se dispone es válido y sirve para este propósito.

Lo que se necesita en este caso es un gran conjunto de datos que cubran todos los frentes y que no se limite a una parte. Dicho de otra forma, se requieren muchos ejemplos lo suficientemente variados como para que la red neuronal aprenda a clasificar de forma general y no quede entrenada a un subconjunto de los datos. Una puntualización sobre este fichero es que está desbalanceado, lo que significa que hay más ejemplos de una actividad que de otra. Esto no supone un problema siempre que al final se valide dicho fichero y se pueda demostrar que los resultados son positivos.

Para comprobar que las instancias disponibles en el fichero de entrenamiento eran suficientes y proporcionaban un buen entrenamiento, se realizaron varias pruebas gracias al programa Weka (17). La principal prueba realizada consistía en una validación cruzada de 10 subconjuntos, lo que permitía determinar que la red entrenaba correctamente y no estaba hecha sólo para aquellas tuplas disponibles en el fichero.

Los resultados que proporciona Weka respecto a la ejecución de una validación cruzada son los siguientes:

- **Kappa** (22): El índice *kappa* corrige el grado de concordancia entre las predicciones del clasificador y la realidad, teniendo en cuenta el número de aciertos que pudieran darse de forma casual.
- **Media del error absoluto**: Es la media de la resta entre el valor obtenido de la clasificación y el valor real del cual se dispone en cada una de las tuplas.
- **Raíz cuadrada del error absoluto**: Al igual que en el apartado anterior, calcula la raíz cuadrada de la resta entre el valor obtenido de la clasificación y el valor real del cual se dispone en cada una de las tuplas.
- **Error absoluto relativo**: Es el cociente entre el error absoluto y el valor exacto. Si se multiplica por 100, se obtiene el tanto por ciento de error.
- **Raíz cuadrada del error absoluto relativo**: Es la raíz cuadrada del cociente entre el error absoluto y el valor exacto.

El resultado de esta validación fue el mostrado en la Tabla 33:

Resultado de validación cruzada de 10 subconjuntos	
Casos clasificadas correctamente	5539 → 96,9373 %
Casos clasificadas incorrectamente	175 → 3,0627 %
Kappa	0,9544
Media del error absoluto	0,0107
Raíz cuadrada del error absoluto	0,0868
Error absoluto relativo	4,7268 %
Raíz cuadrada del error absoluto relativo	25,8263 %

Tabla 33. Resultados de validación cruzada de 10 subconjuntos.

En la Tabla 34 se incluye la matriz de confusión del conjunto de datos utilizado para comprobar los resultados obtenidos, quedando en la columna izquierda la clasificación real y en la parte superior la obtenida por la red de neuronas.

	Estar sentado	Estar de pie	Caminar	Correr	Subir escaleras	Bajar escaleras
Estar sentado	677	0	0	0	0	0
Estar de pie	0	303	5	0	0	0
Caminar	0	0	1707	3	10	2
Correr	0	0	1	2635	0	0
Subir escaleras	5	0	63	4	82	34
Bajar escaleras	1	0	45	0	2	135

Tabla 34. Matriz de confusión de validación cruzada de 10 subconjuntos.

Como se puede comprobar en la Tabla 34, las actividades que peor se clasifican son subir escaleras y bajar escaleras, ya que entre ellas el movimiento del terminal es similar y también puede dar lugar a confusión con la acción de caminar. Por el contrario, correr es una actividad con una precisión casi perfecta debido a la naturaleza de su movimiento, solo superada por la acción de estar sentado y estar de pie. Son patrones de actividad fácilmente reconocibles. Caminar también dispone de una buena clasificación aunque, como ya se ha comentado, a veces se confunde con las acciones de subir y bajar escaleras, así como correr, dependiendo de la velocidad con la que se camine.

6.3.5. Pruebas finales del sistema

En estas pruebas finales se probará ya la aplicación terminada con varios usuarios de variada edad, sexo, etc. Para ello, se les ha entregado uno de los terminales utilizados en las pruebas con la aplicación ya abierta y en la pantalla inicial. Se les ha explicado la finalidad de dicha aplicación, mostrado el manual y solicitado que realizaran un reconocimiento de la actividad que ellos deseasen, pudiendo variarla a lo largo de la prueba.

Los resultados obtenidos se compararán con resultados reales que fueron tomados manualmente en compañía del usuario para obtener el error de clasificación y una breve crítica (ya sea positiva o negativa) al respecto. A continuación se muestran las pruebas realizadas:

	Tiempo real	Tiempo estimado	Desviación (seg)
Correr	00:10:00	00:09:52	8
Caminar	00:20:00	00:19:50	10
Estar de pie	00:05:00	00:05:10	10
Estar sentado	00:10:00	00:10:04	4
Subir escaleras	00:00:00	00:00:03	3
Bajar escaleras	00:00:00	00:00:01	1
Terminal:	Samsung Galaxy SIII – I9300		
Usuario:	Gregorio Martín Berzal – Hombre de 54 años.		
Crítica:	La interfaz es realmente sencilla y no se necesita nada más: Empezar y Parar, y los resultados obtenidos. Como punto positivo la precisión que, aunque no es perfecta, si que se acerca mucho a la realidad. Como punto negativo, añadiría un historial de cada vez que se ejecuta, un total de tiempos… algo así.		

Tabla 35. Prueba final de Gregorio Martín Berzal.

	Tiempo real	Tiempo estimado	Desviación (seg)
Correr	00:05:00	00:05:14	14
Caminar	00:40:00	00:39:46	14
Estar de pie	00:05:00	00:05:20	20
Estar sentado	00:15:00	00:14:50	10
Subir escaleras	00:05:00	00:04:58	2
Bajar escaleras	00:05:00	00:04:52	8
Terminal:	Samsung Galaxy SIII – I9300		
Usuario:	María Nievez de Castro Capontes – Mujer de 50 años.		
Crítica:	Muy fácil de usar y eso que yo de estas cosas no entiendo. Todo se ve muy bien aunque le falta algo de gracia, algún colorcito o algo que le de un toque más encendido jeje. Pero me gusta y funciona muy bien.		

Tabla 36. Prueba final de María Nieves de Castro Capontes.

	Tiempo real	Tiempo estimado	Desviación (seg)
Correr	00:15:00	00:15:08	8
Caminar	00:30:00	00:30:12	12
Estar de pie	00:10:00	00:09:47	13
Estar sentado	00:20:00	00:19:52	8
Subir escaleras	00:04:00	00:03:57	3
Bajar escaleras	00:00:00	00:00:04	4
Terminal:	Samsung Galaxy SIII – I9300		
Usuario:	Sara Carrión Duque – Mujer de 22 años.		
Crítica:	Es muy fea y sosa. Yo le pondría al final, cuando te paras, un muñeco que te resuma las actividades que has realizado, en plan vídeo de corta duración. También algún color que le dé vida a la cosa. Por lo demás *tutto bene*, es muy precisa y todo.		

Tabla 37. Prueba final de Sara Carrión Duque.

	Tiempo real	Tiempo estimado	Desviación (seg)
Correr	00:20:00	00:20:13	13
Caminar	00:30:00	00:29:37	23
Estar de pie	00:08:00	00:07:49	11
Estar sentado	00:03:00	00:03:13	13
Subir escaleras	00:00:00	00:00:05	5
Bajar escaleras	00:00:00	00:00:03	3
Terminal:	LG Optimus Black		
Usuario:	Mario Martín Vizcaíno – Hombre de 22 años.		
Crítica:	La interfaz la veo un poco pobre. Si bien es cierto que, para qué quieres más, pero está muy pobre. Ahora bien, la precisión es muy buena y me ha encantado.		

Tabla 38. Prueba final de Mario Martín Vizcaíno.

	Tiempo real	Tiempo estimado	Desviación (seg)
Correr	00:12:00	00:12:04	4
Caminar	00:15:00	00:14:50	10
Estar de pie	00:15:00	00:14:48	12
Estar sentado	00:30:00	00:29:46	14
Subir escaleras	00:03:00	00:03:07	7
Bajar escaleras	00:01:30	00:01:25	5
Terminal:	Orange Monte Carlo		
Usuario:	Ana Hinojosa Sánchez – Mujer de 23 años.		
Crítica:	Un poco fea la aplicación. Todo blanco y negro parece que hemos vuelto al siglo pasado en colores. Ahora, clasifica muy bien, aunque no separe los tiempos de actividades que no son contiguas pero son la misma. Es decir, correr-andar-correr, te lo junta todo.		

Tabla 39. Prueba final de Ana Hinojosa Sánchez.

Como se puede comprobar y como era de esperar, la precisión no es perfecta. Sin embargo, la desviación no va más allá de unos segundos (por arriba o por abajo) respecto al valor real. Uno de los inconvenientes presentes en esta aplicación (y más concretamente de la red neuronal) es que requiere de una ventana de 20 segundos para realizar su clasificación. Por lo tanto, al cambiar de una actividad a otra, hay momentos que tiene datos de una actividad y datos de la siguiente, por lo que el resultado es algo imprevisible. Sin embargo, no hay una desviación importante y se puede considerar que el modelo cumple con las expectativas.

6.4. Resumen de la evaluación

Gracias a las diferentes pruebas realizadas sobre la aplicación se ha determinado que esta funciona correctamente. El modelo seleccionado para la red de neuronas provee un porcentaje de acierto muy bueno, por lo que se confirma que se adapta al dominio de datos aplicado. Las críticas finales de los usuarios que han tenido contacto y han utilizado la aplicación son positivas en cuanto a la usabilidad y la calidad de los resultados obtenidos, y un poco negativas en cuanto a la interfaz de la aplicación. Las opiniones aportadas por estos usuarios serán tenidas en cuenta en futuras versiones de la aplicación para mejorar la misma.

7. CONCLUSIONES

Para este Trabajo Final de Grado se ha diseñado y desarrollado una aplicación para el sistema operativo Android. Dicha aplicación está destinada al reconocimiento de patrones de actividad del usuario gracias a una red de neuronas artificiales previamente entrenada para tal efecto y a los datos recibidos del acelerómetro disponible en el terminal. La red de neuronas se encarga de clasificar los datos recibidos del acelerómetro y devuelve un valor correspondiente a una de las seis actividades que puede estar realizando el usuario.

Para una mejor organización de la aplicación se eligió programar con la arquitectura Modelo-Vista-Controlador, que permite separar claramente la interfaz, la lógica y los datos del sistema. Debido a que la aplicación estaba destinada para Android, se ha programado en el lenguaje Java que es de alto nivel. Gracias a ello, se han utilizado múltiples características que nos proporciona este lenguaje, como por ejemplo la separación en paquetes de las clases, la visibilidad de los atributos y las funciones disponibles, y la creación de hilos de proceso paralelos.

Por otro lado, aunque la aplicación se desarrolla en Java, no hay que olvidar que está destinada para Android, y este ofrece una amplia API con todas sus funcionalidades y características. Haciendo uso de esta API, se han utilizado *Activities* para la ventana de la aplicación con la que interactúa el usuario y *Managers* para capturar los datos del acelerómetro.

No hay que olvidar Weka, un conjunto de paquetes y librerías escritas en Java que permite realizar multitud de acciones con redes neuronales, ya sea el entrenamiento, la clasificación, etc. Gracias a ello se ha podido construir una red neuronal integrada en el propio terminal que clasifique los datos del acelerómetro y devuelva un solo dato que indique la actividad realizada por el usuario.

Gracias a las pruebas realizadas, se ha comprobado que todo el trabajo realizado hasta ese momento funcionaba correctamente, ya que los errores y desviaciones obtenidos en todas y cada una de ellas era mínimo. Se ha demostrado que es posible ejecutar un proceso tan pesado como es el de una red neuronal, en terminales con recursos limitados como es un *smartphone*. Además, los usuarios finales a los que se les dejo probar el sistema, están realmente satisfechos con el resultado final.

Para finalizar y en relación con los objetivos del proyecto, este trabajo aporta como novedad dentro del ámbito de aplicaciones destinadas a *smartphones* el uso y explotación de redes neuronales ligeras para el reconocimiento de patrones de actividad. Gracias a la recogida de multitud de datos (mediante otras aplicaciones) y al entrenamiento previo de la red neuronal, se ha obtenido un sistema de reconocimiento de patrones de actividad que se ejecuta dentro del tiempo establecido y permite una clasificación casi en tiempo real.

8. LÍNEAS FUTURAS

En este capítulo se planten líneas de desarrollo que pueden ser estudiadas y llevadas a cabo en un futuro. Entre estas futuras líneas, se proponen las siguientes:

- **Aumentar los patrones de actividad reconocidos**. Como se ha explicado anteriormente, el sistema es capaz de captar 6 patrones de actividad diferentes. Sin embargo, una persona puede realizar muchos más patrones, como por ejemplo montar en bicicleta o saltar. Esta ampliación de la aplicación es bastante simple ya que solo se requiere para ello un modelo de entrenamiento mayor para que la red de neuronas lo aprenda. El resto de la aplicación permanece inalterado.

- **Portar la aplicación a otros sistemas operativos**. Como bien se ha dicho, Android es uno de los más comunes sistemas operativos, pero no es el único. iOS, Windows Phone, BlackBerry y Symbian son más sistemas que se pueden encontrar en los *smartphones*. Para que la aplicación funcione, tan solo se requiere que el terminal disponga de un acelerómetro de tres ejes.

- **Nuevos modelos de clasificación**. En la aplicación aquí desarrollada se ha utilizado el perceptrón multicapa como modelo de clasificación de los datos. Un estudio más exhaustivo de otros modelos podría revelar alguno con mayor precisión, velocidad o incluso entrenamiento continuado, es decir, que el usuario pueda entrenar el clasificador de su aplicación. Si se encontrara un modelo mejor, simplemente habría que publicar la nueva versión en las tiendas online de cada sistema operativo para que los usuarios descargasen la nueva actualización.

- **Conexión con redes sociales**. Conectar una aplicación con las redes sociales está muy de moda hoy en día, sobre todo para publicar qué está haciendo el usuario o el resultado de alguna acción. En esta aplicación, mostrar las actividades que ha realizado el usuario puede ser muy interesante, ya sea simplemente por compartir sus datos o para competir con familiares y amigos. Las principales redes sociales que deberían agregarse son Facebook y Twitter.

- **Internacionalización de la aplicación**. Android, al igual que el resto de sistemas, es un sistema operativo a nivel mundial y, como tal, debe estar disponible en el idioma de la región. Aunque actualmente solo está disponible en español, la aplicación habría que traducirla a multitud de lenguajes como por ejemplo inglés y chino, que son los más utilizados en el planeta.

GLOSARIO

- **Acelerómetro**: Aparato destinado a medir las aceleraciones. Combinado con la gravedad, permite, en los 3 ejes, determinar la posición del acelerómetro y, por ende, la del terminal que lo incluya

- **Activity**: Clase de la API de Android que representa a una ventana de la aplicación, incluyendo sus opciones, menús, botones y demás.

- **API**: Interfaz de Programación de Aplicaciones (*Application Programming Interface*). Es una serie de librerías que ofrecen al usuario la posibilidad de utilizar el potencial disponible de una herramienta o sistema, como puede ser Android.

- **APK**: *Application PacKage file*. Es la extensión y el formato de los ficheros de Android que contienen aplicaciones para el sistema operativo.

- **ARM**: *Advanced RISC Machine*. Es una arquitectura de procesadores de 32bits desarrollada por la empresa ARM Holdings.

- **GPS**: Sistema de Posicionamiento Global (*Global Positioning System*). Es un sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto con una precisión hasta de centímetros, aunque lo habitual son unos pocos metros de precisión.

- **Hardware**: Partes tangibles de un sistema informático, como el procesador, la pantalla, la memoria, los botones…

- **Interfaz**: es el medio con que el usuario puede comunicarse con una máquina y comprende todos los puntos de contacto entre el usuario y el equipo.

- **Java**: Lenguaje de programación orientado a objetos desarrollado en 1995 y que se utiliza para programar las aplicaciones del sistema operativo Android.

- **Kernel**: Es el núcleo del sistema operativo y la parte más importante del mismo, ya que se encarga, principalmente, del acceso seguro a los componentes hardware disponibles.

- **Log**: Registro de eventos durante un rango de tiempo en particular. Se utiliza principalmente para registrar datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre para un dispositivo en particular o aplicación.

- **Memoria RAM**: Memoria de Acceso Aleatorio (*Random Memory Access*). Se utiliza como memoria de trabajo para el sistema operativo y se cargan en ella todas las instrucciones que ejecuta el procesador.

- **Multitáctil**: Es una tecnología parte de la interacción persona-computador que permite reconocer varios puntos de contacto simultáneamente en una pantalla táctil.

- **Neurona**: Unidad de cálculo que intenta emular el comportamiento de una neurona natural, como las del cerebro humano. Unidas, constituyen una red de neuronas artificiales.

- **Paquete**: Contenedor de clases de Java que permite agrupar las distintas partes de un programa cuya funcionalidad tiene elementos comunes.
- **Patrón de actividad**: Acción que está realizando el usuario y el patrón que la representa, en este caso, el movimiento necesario para dicha actividad.
- **Plug-in**: Complemento o aplicación que se relaciona con otra superior para aportarle una nueva funcionalidad.
- **Red de neuronas artificial**: Formado por varias neuronas (artificiales), pretende emular una red de neuronas real como la del cerebro humano, capaz de clasificar elementos a partir de algunos datos de los mismos. Estas redes artificiales se utilizan en programación.
- **SDK**: Kit de Desarrollo de Software (*Software Development Kit*). Es un conjunto de herramientas de desarrollo de software que permite al programador crear aplicaciones para un sistema concreto de forma más sencilla.
- **Sistema operativo**: Programa o conjunto de programas que, en un sistema informático, gestiona los recursos de hardware y provee servicios a los programas de aplicación.
- **Smartphone**: Teléfono inteligente construido sobre una plataforma informática móvil con mayor capacidad y conectividad que los teléfonos móviles convencionales.
- **Software**: Parte lógica de un sistema informático que comprende el conjunto de componentes lógicos necesarios que hacen posible la realización de tareas específicas.
- **Thread**: Clase de la API de Java que permite crear un hilo de ejecución en paralelo para ser ejecutado en cualquier momento, sin necesidad de esperar a que finalice una ejecución anterior.
- **TimerTask**: Clase de la API de Java que permite ejecutar *Threads* de forma cíclica cada X tiempo.
- **UML**: Lenguaje Unificado de Modelado (*Unified Modeling Language*). Es el lenguaje de modelado de sistemas software más conocido y utilizado, que representa de forma gráfica un sistema.
- **Usuario**: Individuo que utiliza el dispositivo y realiza las múltiples operaciones del sistema con distintos propósitos.
- **XML**: Lenguaje de Marcas eXtensible (*eXtensible Markup Language*). Es un lenguaje de marcas que permite definir la gramática de lenguajes específicos para estructurar documentos grandes.

REFERENCIAS

1. Android. *Wikipedia, la enciclopedia libre.* [En línea] 31 de Marzo de 2012. http://es.wikipedia.org/wiki/Android.

2. **Gartner.** Gartner afirma que Android llegará al 50% de cuota de mercado en 2012. *Gartner Newsroom.* [En línea] 7 de Abril de 2011. http://www.gartner.com/it/page.jsp?id=1622614.

3. **Google Inc.** Google Play. [En línea] 2012. https://play.google.com/.

4. Acelerómetro. *Wikipedia, la enciclopedia libre.* [En línea] 5 de Enero de 2012. http://es.wikipedia.org/wiki/Aceler%C3%B3metro.

5. **Rivero Hernández, Diego.** Diseño y simulación de un enrutador óptico con dispositivos de conmutación basados en resonadores en anillo. *Archivo Abierto Institucional de la Universidad Carlos III de Madrid.* [En línea] Diciembre de 2008. http://hdl.handle.net/10016/5518.

6. **Cano Monedero, Ángel.** Algoritmos genéticos para la resolución del problema de BCI. *Archivo Abierto Institucional de la Universidad Carlos III de Madrid.* [En línea] 15 de Diciembre de 2011. http://hdl.handle.net/10016/13293.

7. **Ortiz de Lazcano Lobato, Juan Miguel.** El perceptrón simple. *Lenguages y Ciencias de la Computación. Universidad de Málaga.* [En línea] 12 de Marzo de 2004. http://www.lcc.uma.es/~jmortiz/archivos/Tema4.pdf.

8. Perceptrón multicapa. *Wikipedia, la enciclopedia libre.* [En línea] 15 de Marzo de 2012. http://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa.

9. **Moreno Rodríguez, Alfonso.** Desarrollo de una interfaz gráfica de redes neuronales usando Matlab. *Archivo Abierto Institucional de la Universidad Carlos III de Madrid.* [En línea] 2009. http://hdl.handle.net/10016/8488.

10. Validación cruzada. *Wikipedia, la enciclopedia libre.* [En línea] 4 de Junio de 2012. http://es.wikipedia.org/wiki/Validación_cruzada.

11. **SportsTrackLive.** Sports Tracker Pro. *Google Play.* [En línea] 28 de Febrero de 2012. https://play.google.com/store/apps/details?id=com.sportstracklive.android.ui.activity.pro.

12. **runastic.** runastic Pro. *Google Play.* [En línea] 26 de Marzo de 2012. https://play.google.com/store/apps/details?id=com.runtastic.android.pro2.

13. **Viaden Gaming Limited.** All-in Podómetro +. *Google Play.* [En línea] 16 de Febrero de 2012. https://play.google.com/store/apps/details?id=com.viadengambling.pedometer.full.

14. **Arawella Corporation.** Podómetro PRO GPS +. *App Store.* [En línea] 23 de Diciembre de 2011. http://itunes.apple.com/es/app/pedometer-pro-gps-+/id388383179.

15. **Smartrunner GmbH.** SmartRunner. *Marketplace de Windows Phone.* [En línea] 20 de Octubre de 2010. http://www.windowsphone.com/es-ES/apps/34f24984-cdda-df11-a844-00237de2db9e.

16. **Eclipse Foundation.** Eclipse. [En línea] 2012. http://www.eclipse.org/.

17. **Hall, Mark, y otros, y otros.** The WEKA Data Mining Software. [En línea] 2009. http://www.cs.waikato.ac.nz/ml/weka/.

18. **rjmarsan.** Weka for Android. [En línea] 16 de Febrero de 2011. https://github.com/rjmarsan/Weka-for-Android.

19. **GitHub Inc.** GitHub - Social Coding. [En línea] 2012. https://github.com/.

20. **Google Inc.** ADT Plugin. [En línea] Agosto de 2012. http://developer.android.com/tools/sdk/eclipse-adt.html.

21. **Fowler, Martin y Scott, Kendall.** *UML Gota a gota.* 2000.

22. **Cohen, Jacob.** A coefficient of agreement for nominal scales. *Educational and Psychological Measurement.* 1960, Vol. 20, págs. 37-46.

23. **Tomás Cadenas, José.** Diagramas de Gantt. [En línea] Mayo de 2009. http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r56714.PDF.

ANEXO I: MANUAL DE USUARIO

En este anexo se presenta un breve tutorial sobre el manejo de la aplicación, en el que se incluyen instrucciones de cómo ejecutarla y configurar los parámetros de la misma.

Requisitos

Los requisitos mínimos, al igual que los recomendados, para poder utilizar correctamente esta aplicación son los siguientes:

- *Smartphone* con sistema operativo Android 2.3 o superior.
- Acelerómetro integrado en el terminal.

Instalación

Para la instalación de esta aplicación en el terminal, se pueden optar por dos caminos descritos a continuación:

- A través de la tienda oficial de Android conocida como Google Play. Dentro de la misma tienda, ya sea en el teléfono o a través de la web, se puede buscar la aplicación a través de su identificador "es.uc3m.caos.tfg" o a través del nombre "TFG". Una vez encontrada se pulsa sobre el botón "Instalar" y luego sobre "Aceptar y descargar" para aceptar los permisos que esta aplicación solicita en el terminal.

- En caso de disponer del fichero de instalación APK, se puede instalar manualmente. Para ello, lo primero es habilitar la instalación de aplicaciones desde fuentes desconocidas, opción disponible en el menú de *Ajustes* > *Seguridad* > *Fuentes desconocidas*. Luego, se debe copiar dicho fichero a la memoria del teléfono (ya sea interna o tarjeta de memoria). Después, mediante un explorador de archivos, se localiza el fichero y se pulsa sobre él. El teléfono preguntará si deseamos instalar la aplicación, a lo que se contesta afirmativamente.

Ejecución

Para ejecutar la aplicación, simplemente se debe acceder a ella y pulsar sobre el botón de "Empezar". Una vez se muestre que la aplicación está inicializando, el usuario debe colocar el teléfono en su bolsillo y realizar las actividades que desee. El terminal puede ser bloqueado para evitar que se pulse sobre la pantalla de forma accidental.

Una vez finalice sus actividades, se extrae el terminal del bolsillo y pulsa sobre el botón "Detener" para que el reconocimiento de actividades finalice y pueda observar los resultados obtenidos. Si el usuario vuelve a pulsar sobre "Empezar", el contador de tiempo de los resultados se reiniciará.

Para salir de la aplicación, se debe pulsar sobre el botón *Atrás* disponible en el mismo terminal (no es parte de la aplicación). Una vez pulsado se le pedirá al usuario que confirme su acción de salir de la aplicación, continuando con el proceso actual en segundo plano. En caso de responder "No", la aplicación prosigue normalmente. En caso de responder "Si", se detiene el reconocimiento y se sale de la aplicación.

ANEXO II: PLANIFICACIÓN

En este apartado del proyecto se muestra la planificación que se ha ido realizando durante el desarrollo de la aplicación. Para ello se utiliza el diagrama de Gantt (23), el cual muestra el tiempo de dedicación previsto y real para las diferentes tareas o actividades a lo largo del tiempo de desarrollo.

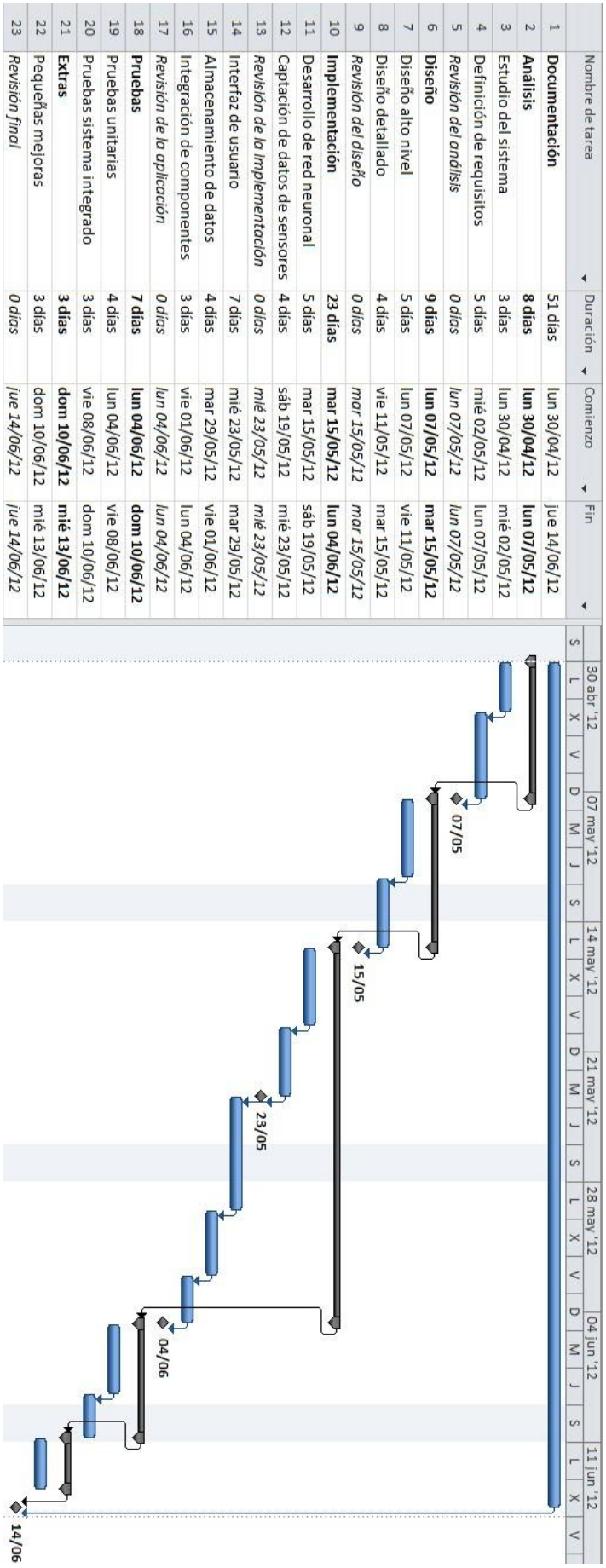
Para analizar la planificación seguida, se van a mostrar tres diagramas diferentes:

- El diagrama de la Ilustración 14 muestra la planificación ideal cuando se inició el proyecto:
- El diagrama de la Ilustración 15 muestra el tiempo real de dedicación en el desarrollo del sistema.
- El diagrama de la Ilustración 16 muestra una comparativa entre la planificación inicial y el tiempo real empleado en el desarrollo, mostrándose la desviación temporal existente.

Tal y como se ve en los diagramas de Gantt, ha habido una pequeña desviación tanto en tiempo como en el orden de realización de cada una de las etapas. Esto es debido a que la planificación inicial estaba basada en un modelo ideal de planificación, pero esta idea inicial no fue posible, ya que se descubrió que el desarrollo sería más sencillo si se iniciaba primero el apartado de la memoria, debido a que se empezaría a tomar contacto con el lenguaje de programación y el entorno de desarrollo del proyecto.

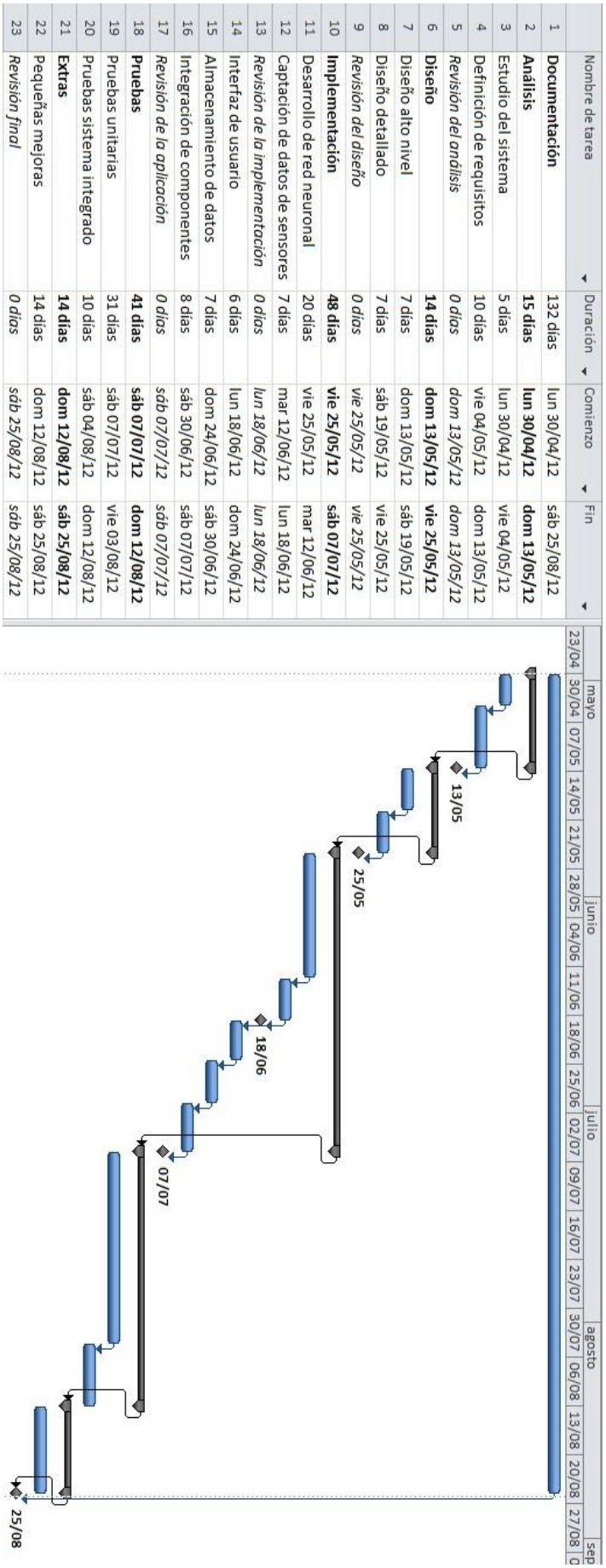
El retraso también se ha visto influido positivamente por la capacidad de perfeccionismo del equipo de desarrollo, ya que han trabajado duramente gastando algo más del tiempo de desarrollo para obtener una aplicación con mejor calidad y velocidad de ejecución.

Por último, comentar que este desfase en el desarrollo no ha influido en el trabajo final, ya que lo que desde un principio se deseaba era finalizarlo antes del fin del año lectivo, como finalmente ha sido.



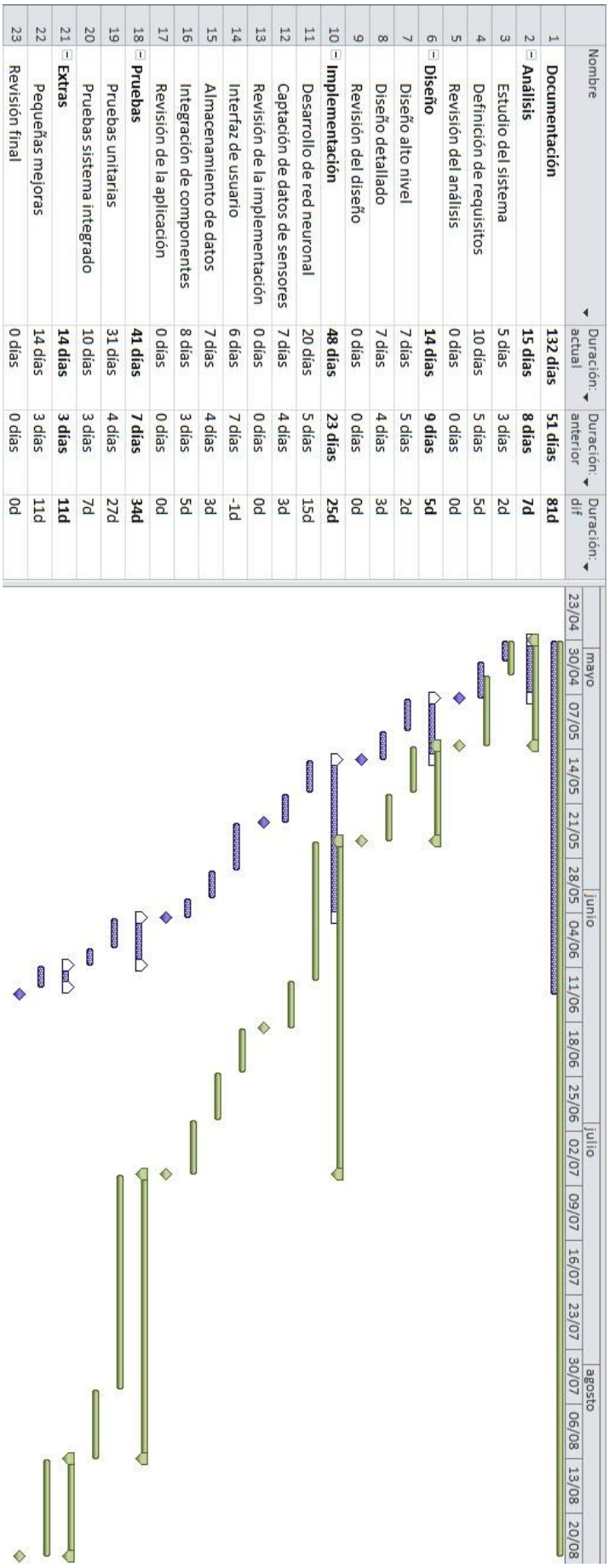
	Nombre de tarea	Duración	Comienzo	Fin
1	**Documentación**	51 días	lun 30/04/12	jue 14/06/12
2	**Análisis**	**8 días**	**lun 30/04/12**	**lun 07/05/12**
3	Estudio del sistema	3 días	lun 30/04/12	mié 02/05/12
4	Definición de requisitos	5 días	mié 02/05/12	lun 07/05/12
5	*Revisión del análisis*	*0 días*	*lun 07/05/12*	*lun 07/05/12*
6	**Diseño**	**9 días**	**lun 07/05/12**	**mar 15/05/12**
7	Diseño alto nivel	5 días	lun 07/05/12	vie 11/05/12
8	Diseño detallado	4 días	vie 11/05/12	mar 15/05/12
9	*Revisión del diseño*	*0 días*	*mar 15/05/12*	*mar 15/05/12*
10	**Implementación**	**23 días**	**mar 15/05/12**	**lun 04/06/12**
11	Desarrollo de red neuronal	5 días	mar 15/05/12	sáb 19/05/12
12	Captación de datos de sensores	4 días	sáb 19/05/12	mié 23/05/12
13	*Revisión de la implementación*	*0 días*	*mié 23/05/12*	*mié 23/05/12*
14	Interfaz de usuario	7 días	mié 23/05/12	mar 29/05/12
15	Almacenamiento de datos	4 días	mar 29/05/12	vie 01/06/12
16	Integración de componentes	3 días	vie 01/06/12	lun 04/06/12
17	*Revisión de la aplicación*	*0 días*	*lun 04/06/12*	*lun 04/06/12*
18	**Pruebas**	**7 días**	**lun 04/06/12**	**dom 10/06/12**
19	Pruebas unitarias	4 días	lun 04/06/12	vie 08/06/12
20	Pruebas sistema integrado	3 días	vie 08/06/12	dom 10/06/12
21	**Extras**	**3 días**	**dom 10/06/12**	**mié 13/06/12**
22	Pequeñas mejoras	3 días	dom 10/06/12	mié 13/06/12
23	*Revisión final*	*0 días*	*jue 14/06/12*	*jue 14/06/12*

Ilustración 14. Diagrama de Gantt correspondiente a la planificación inicial del proyecto.



	Nombre de tarea	Duración	Comienzo	Fin
1	**Documentación**	132 días	lun 30/04/12	sáb 25/08/12
2	**Análisis**	**15 días**	**lun 30/04/12**	**dom 13/05/12**
3	Estudio del sistema	5 días	lun 30/04/12	vie 04/05/12
4	Definición de requisitos	10 días	vie 04/05/12	dom 13/05/12
5	*Revisión del análisis*	*0 días*	*dom 13/05/12*	*dom 13/05/12*
6	**Diseño**	**14 días**	**dom 13/05/12**	**vie 25/05/12**
7	Diseño alto nivel	7 días	dom 13/05/12	sáb 19/05/12
8	Diseño detallado	7 días	sáb 19/05/12	vie 25/05/12
9	*Revisión del diseño*	*0 días*	*vie 25/05/12*	*vie 25/05/12*
10	**Implementación**	**48 días**	**vie 25/05/12**	**sáb 07/07/12**
11	Desarrollo de red neuronal	20 días	vie 25/05/12	mar 12/06/12
12	Captación de datos de sensores	7 días	mar 12/06/12	lun 18/06/12
13	*Revisión de la implementación*	*0 días*	*lun 18/06/12*	*lun 18/06/12*
14	Interfaz de usuario	6 días	lun 18/06/12	dom 24/06/12
15	Almacenamiento de datos	7 días	dom 24/06/12	sáb 30/06/12
16	Integración de componentes	8 días	sáb 30/06/12	sáb 07/07/12
17	*Revisión de la aplicación*	*0 días*	*sáb 07/07/12*	*sáb 07/07/12*
18	**Pruebas**	**41 días**	**sáb 07/07/12**	**dom 12/08/12**
19	Pruebas unitarias	31 días	sáb 07/07/12	vie 03/08/12
20	Pruebas sistema integrado	10 días	sáb 04/08/12	dom 12/08/12
21	**Extras**	**14 días**	**dom 12/08/12**	**sáb 25/08/12**
22	Pequeñas mejoras	14 días	dom 12/08/12	sáb 25/08/12
23	*Revisión final*	*0 días*	*sáb 25/08/12*	*sáb 25/08/12*

Ilustración 15. Planificación real del proyecto al finalizar el mismo.



	Nombre	Duración: actual	Duración: anterior	Duración: dif
1	**Documentación**	**132 días**	**51 días**	**81d**
2	**Análisis**	**15 días**	**8 días**	**7d**
3	Estudio del sistema	5 días	3 días	2d
4	Definición de requisitos	10 días	5 días	5d
5	Revisión del análisis	0 días	0 días	0d
6	**Diseño**	**14 días**	**9 días**	**5d**
7	Diseño alto nivel	7 días	5 días	2d
8	Diseño detallado	7 días	4 días	3d
9	Revisión del diseño	0 días	0 días	0d
10	**Implementación**	**48 días**	**23 días**	**25d**
11	Desarrollo de red neuronal	20 días	5 días	15d
12	Captación de datos de sensores	7 días	4 días	3d
13	Revisión de la implementación	0 días	0 días	0d
14	Interfaz de usuario	6 días	7 días	-1d
15	Almacenamiento de datos	7 días	4 días	3d
16	Integración de componentes	8 días	3 días	5d
17	Revisión de la aplicación	0 días	0 días	0d
18	**Pruebas**	**41 días**	**7 días**	**34d**
19	Pruebas unitarias	31 días	4 días	27d
20	Pruebas sistema integrado	10 días	3 días	7d
21	**Extras**	**14 días**	**3 días**	**11d**
22	Pequeñas mejoras	14 días	3 días	11d
23	Revisión final	0 días	0 días	0d

Ilustración 16. Comparativa entre ambas planificaciones.

ANEXO III: PRESUPUESTO

En este anexo del proyecto se muestra el presupuesto para el desarrollo del sistema, en el que se hace una descripción detallada de los gastos del proyecto, incluyendo el personal que ha desarrollado el sistema y los equipos utilizados para ello. Además, se ha hecho un estudio en el que se analiza el número de ventas mínimo para amortizar el desarrollo de este proyecto. El presupuesto detallado se encuentra en la Ilustración 17.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
David Martín de Castro

2.- Departamento:
Ciencias de la Computación e Inteligencia Artificial

3.- Descripción del Proyecto:

- Título	**Trabajo Final de Grado: Redes neuronales en terminales smartphone para el reconocimiento de patrones de actividad**
- Duración (meses)	**4**

5.- Desglose presupuestario (costes directos)

PERSONAL

Categoría	Dedicación (hombres mes) [a)]	Coste hombre hora	Coste hombre mes	Coste (Euro)
Jefe de Proyecto	130	30	3.900,00	15.600,00
Analista	120	25	3.000,00	12.000,00
Programador	140	15	2.100,00	8.400,00
Encargado de pruebas	100	15	1.500,00	6.000,00
			Total	**42.000,00**

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Acer Aspire 5740DG	800,00	100	4	60	53,33
Samsung Galaxy SIII	700,00	50	4	60	23,33
LG Optimus Black	400,00	50	4	60	13,33
Orange Monte Carlo	375,00	25	4	60	6,25
				Total	**96,25**

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

6.- Resumen de costes

	Presupuesto Costes Totales
Personal	42.000,00 €
Amortización	96,25 €
Total	**42.096,25 €**

7.- Amortización del producto

Coste total proyecto	Coste inscripción Google Play	Precio aplicación	Porcentaje Google Play	Beneficio por venta	**Total Ventas**
42.096,25 €	25,00 €	2,99 €	30%	2,09 €	**20.125**

Ilustración 17. Presupuesto del Trabajo Final de Grado.